Adaptive Behavior with Fixed Weights in RNN: An Overview*

Danil V. Prokhorov[$], Lee A. Feldkamp[$] and Ivan Yu. Tyukin[+]
[$]Ford Research Laboratory, Dearborn, MI 48121, U.S.A.
[+]Saint-Petersburg State Electrotechical University, Russia, and
RIKEN Brain Science Institute, Japan

Abstract

In this paper we review recent results on adaptive behavior attained with fixed-weight recurrent neural networks (meta-learning). We argue that such behavior is a natural consequence of prior training.

1 Introduction

Emergence of adaptive behavior from a recurrent neural network (RNN) with fixed weights has been noticed by various authors (see, e.g., [1], [2], [3] and [4]). While the ability to adapt to a changed environment is conventionally attributed to systems whose parameters change in response to an environmental change, a fixed-weight RNN can acquire such an ability through prior training or, sometimes, by construction. This happens because an RNN possesses internal recurrence, so there is no need to change its weights to react to a changing environment.

Different researchers denote the adaptive behavior of RNN differently. It is termed meta-learning (learning how to learn) in [5], whereas the name accommodative is suggested in [4].

This paper consists of three sections. In the next section we briefly review recent results on meta-learning. Section 3 describe two illustrative problems and their solutions with recurrent multilayer perceptrons (RMLP), followed by discussion in Section 4. We also show evolution of outputs of recurrent nodes in RMLP. We conclude in Section 5 with comments on future research.

2 Overview

Recent experiments on meta-learning with fixed-weight RNN deal with two broad classes of problems. Class I encompasses neural approximation of multiple input-output mappings of the following form

$$\mathbf{y}^d(t) = \mathbf{f}_\theta(\mathbf{z}_\theta(t-1), \mathbf{x}_\theta(t)) \quad (1)$$

where $\mathbf{f}_\theta$ is a discrete or continuous set of mappings with the output vector $\mathbf{y}^d(t)$ at time t, $\mathbf{x}_\theta$ is a vector of inputs, and $\mathbf{z}_\theta$ is the mapping's state vector (evolution of $\mathbf{z}_\theta$ may be represented by a separate equation which is avoided in our notation as it is assumed to be a part of $\mathbf{f}_\theta$). The RNN approximating $\mathbf{y}^d$ for all t in the mean square sense has the form

$$\hat{\mathbf{y}}(t) = \mathbf{f}(\mathbf{z}(t-1), \mathbf{x}_\theta(t)) \quad (2)$$

where $\mathbf{z}$ is its state vector. Sometimes none of the mappings have states $\mathbf{z}_\theta$, as in [3], [5] and [6]. Furthermore, the input $\mathbf{x}_\theta(t)$ may include the previous value of the target output $\mathbf{y}^d(t-1)$ to provide the network with appropriate context.

Class II includes problems in which accurate control of multiple distinct systems $\mathbf{g}_\theta$ (or plants) is required:

$$\hat{\mathbf{y}}(t) = \mathbf{g}_\theta(\mathbf{z}_\theta(t-1), \mathbf{f}(\hat{\mathbf{y}}(t-1), \mathbf{z}(t-1), \mathbf{x}_\theta(t))) \quad (3)$$

Here the system's output $\hat{\mathbf{y}}(t)$ should closely track the target output $\mathbf{y}^d(t)$ produced by a reference model (e.g., $\mathbf{y}^d(t)$ can be zero at all times, as in [2]). The input $\hat{\mathbf{y}}(t-1)$ of the controller RNN $\mathbf{f}$ may or may not include $\mathbf{z}_\theta(t-1)$ (or part thereof). Another input $\mathbf{x}_\theta(t)$ includes $\mathbf{y}^d(t)$ and, possibly, other external signals.

In [3], structured RNN are proposed to model the given set of mappings of (1). Such RNN include not only parts of networks that approximate the desired mappings but also learning algorithms. One such structure for a problem of approximating *all* quadratic functions of two variables is shown in Figure 1. It can be seen that recurrent connections (nodes for a, b, c, d, e and f) have a feedback weight of unity, and their adaptation is governed by the past derivatives $\partial y(t-1)/\partial a(t-1)$, $\partial y(t-1)/\partial b(t-1)$, etc. The parameter α acts as a learning rate which can be fixed to a small value or learned in a training session (recall that the network weights must be fixed during its operation; their role is taken by the states a, b, etc.) The network of Figure 1 can be represented by an RNN of general architecture consisting of summation and product nodes with delayed connections.

In [5], a special form of RNN called *long short-term memory* (LSTM) is explored. In one of its modules the LSTM has the unity feedback weights which are claimed

*The first author is pleased to acknowledge a helpful correspondence with Dr. Steven Younger.

0-7803-7278-6/02/$10.00 ©2002 IEEE

Authorized licensed use limited to: Akira Imada. Downloaded on May 16, 2009 at 19:53 from IEEE Xplore. Restrictions apply.

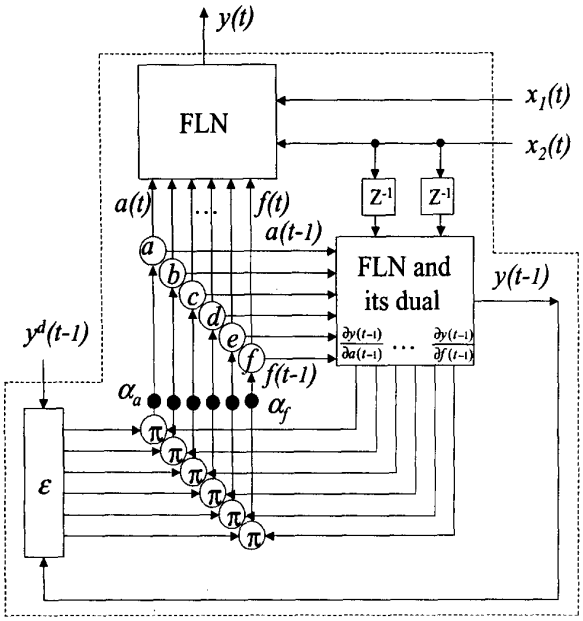

Figure 1: Structured RNN that is capable of learning all quadratic functions of two variables. It is enclosed within the dashed contour. FLN stands for functional link network implementing the function $y = ax_1{}^2 + bx_2{}^2 + cx_1x_2 + dx_1 + ex_2 + f$. Each recurrent node, e.g., node a, evolves according to the rule $a(t) = a(t-1) + \alpha_a \epsilon(t-1)\partial y(t-1)/\partial a(t-1)$, where $\epsilon(t-1) = y^d(t-1) - y(t-1)$.

to be needed for efficient training of its remaining weights for several different meta-learning tasks including the one just discussed.

Recent experiments with RMLP for meta-learning suggest that resorting to either structured RNN or LSTM is not necessary. In [1], a single RMLP with three fully recurrent hidden layers (21 states) is trained to make good one-time-step predictions of 13 different time series (periodic and chaotic). The fixed-weight RMLP is demonstrated to be capable of good generalization to time series with somewhat different sets of generating parameters as well as to those corrupted by noise. In [7], achieving good one-time-step predictions of five different time series from a two-hidden layer RMLP (14 states) via training is combined with two conditioning tasks. The trained network must remember which of the two tasks it dealt with in the past (Henon maps, type 1 or 2) in order to activate one of the two appropriate output responses for the random input. All the problems above belong to class I.

In [2], a two-hidden-layer RMLP (20 states) is trained to act as stabilizing controller for three distinct and unrelated systems, without explicit knowledge of system identity. In [8], training an RMLP with 10 states is accomplished to achieve robust control of more than 10,000 systems derived from a single nominal system by parametric perturbations. These problems are examples of (3) and belong to class II.

3 Experiments

The training method used in all the tasks above is based on backpropagation through time (BPTT) and the multi-stream extended Kalman filter algorithm; see [9] for details. Here we discuss two class I meta-learning tasks described in [5] and propose their solutions with RMLP.

The problem of learning all quadratic functions of two variables introduced above is successfully solved by training a RMLP with three inputs, $x_1(t)$, $x_2(t)$ and $y^d(t-1)$, 30 bipolar sigmoid nodes in the first fully recurrent layer, 10 bipolar sigmoid nodes in the second fully recurrent layer and a linear output node $\hat{y}(t)$. Such an RMLP architecture is denoted as 3-30R-10R-1L and has 1441 trainable weights. The inputs and the output are scaled to be approximately within the range ± 1.0. One epoch of training consists of the following steps. First, we randomly choose 20 segments of 1040 consecutive points each within the time series of 128,000 points (128 different quadratic functions of 1000 examples each). The initial 40 points of each segment are used to let the network develop its states (*priming* operation) from their initial states of zeros, rather than for training weights. Next, we apply the 20-stream global EKF to update weights, with derivatives being computed by BPTT with truncation depth of 40 (denoted as BPTT(40)). We use 20×1000 points for training in each epoch. Our training session lasts for 1620 epochs. The first 600 epochs are carried out with the parameter $R_0 = 100$ and the parameter $Q = 0.01$. The process noise Q is decreased to 0.003 and 0.001 at epoch numbers 601 and 1401, respectively. The root mean square (RMS) error attained after 600 epochs of training is equal to 0.0273, and it is equal to 0.020 by the end of training. The final network is tested on two new time series 128,000 points long (examples of totally new quadratic functions) resulting in RMS errors of 0.023 and 0.024.

The problem of learning all 16 Boolean functions of two variables was introduced in [3]. As in the previous task, we use a 3-16R-16R-1 RMLP with three inputs and 865 trainable weights. The inputs and the target output are equal to ± 1.0. The training process is carried out using 16-stream global EKF with BPTT(32), each segment's length of 102 points with only two points at the segment's beginning assigned to priming from random initial states, and the training time series composed of 256 randomly chosen (out of 16) Boolean functions of 256 examples each. We use 16×100 points for training in each epoch. Our training session lasts for 2400 epochs with the same parameters

Authorized licensed use limited to: Akira Imada. Downloaded on May 16, 2009 at 19:53 from IEEE Xplore. Restrictions apply.

as in the quadratic function problem. At the end of training we attain an RMS error of 0.162 with 444 sign errors. The final network is then tested for two new time series representing the same 16 Boolean functions but whose order (of functions themselves and their examples) is different from the one used for training. The test results are an RMS error of 0.178 with 555 sign errors and an RMS error of 0.175 with 533 sign errors, as compared to 643 classification errors for the network in [6][1]. It is important to note that for this and other classification tasks superior values of RMS errors are not as critical as lower counts of errors.

4 Discussion

Our results for these two problems compare favorably to the results for the same problems presented in [6]. Yet, we use the standard RMLP architecture proven to work for other problems. These RMLP are trained to minimize a quadratic function of error between the target output and the output of the network. It should be emphasized that, while the error function is an explicit function of the output, it is also an implicit function of RNN states and, of course, weights. The states are initialized to some values (usually zeros). After initialization they act as dependent variables of the weights.

By virtue of training RNN weights (or, in limited instances, its construction), the evolution of states $\mathbf{z}$ is restricted to specific families of trajectories (orbits). When an RNN senses a particular type of input for which it was trained, its states react so as to produce the output response appropriate for the given input. When a new (but also known to the RNN) type of input is provided, the states switch from one family of orbits to another family which corresponds to the new type. Switching results in an initial transient behavior manifesting itself in a relatively large level of output error that persists for a few data points. When states stabilize at their new orbits, output errors reach a steady state level. This is acceptably small for a well trained RNN, but it is probably impossible to guarantee that errors larger than the steady state may not occasionally occur. In fact, we were able to find such errors in the Boolean problem and they are included in the total count of errors reported here. Further testing on much longer time series did not result in a substantial increase of the error count. For example, testing our Boolean network on 16 time series representing 100,000 randomly chosen examples of each function resulted in less than 1 error per 1000 examples on average.

Evolution of states $\mathbf{z}$ driven by inputs and constrained by the network's architecture and trained weights imitates adaptation of parameters in a conventional adaptive system. It is this evolution that is responsible for emergence of adaptive behavior in RNN with fixed weights. It should be emphasized that there are no requirements for special structures for such RNN, e.g., like those in [3], [5], [6]. (There is no linear feedback with a weight of unity in the standard RMLP architecture, because all recurrent nodes are nonlinear.) Furthermore, it appears possible to extend the results of theoretical analysis in [10], which treats the ability of a single network with output-to-input recurrence to approximate multiple systems to the case of RMLP.

To illustrate the evolution of states, we choose the RMLP of [7] because it has only 14 hidden nodes in its two fully recurrent layers. Figures 2 and 3 show outputs of nodes of both hidden layers and the corresponding output of the network for each segment of the composite time series (the network was previously trained to approximate well five different behaviors shown as individual segments of the time series). Careful examination reveals that each node evolves along a different orbit depending on the segment of the time series. Orbits appear to be not very sensitive to variations in the input signal. Indeed, Figures 4 and 5 show the difference between orbits of each node for the same network in two experiments. In the first experiment the network is fed by the same inputs as in [7]. In the second experiment the network is fed by the inputs corrupted by uniform noise in the range $[-0.05, +0.05]$. Such experiments were repeated many times for different realizations of noise to test the sensitivity of the nodal orbits. The results are similar to those shown in Figures 4 and 5.

5 Open issues

Careful application of powerful training methods such as the one mentioned here enables training RNN for tasks which require adaptive capabilities. Though applied to training RMLP, the training method referred to can be extended straightforwardly to all differentiable RNN, including LSTM. However, several open issues still remain for future research.

1. How to achieve efficient training? While we succeeded in all meta-learning problems attempted thus far using the training method based on BPTT and EKF, the training session for some problems (e.g., quadratic functions) took more than three weeks on 800 MHz PC. Does a more efficient method even exist?
2. How to guarantee long-term stability of solutions? For example, in the two tasks discussed in Section 3 we were able to confirm an acceptable retention of solutions in limited testing the two RMLP on sequences of examples of functions many times longer than those used in training (similar confirmation was made in [7]). But it is plausible that, for some input sequences, any trained RNN can

[1]The errors for the network in [6] were counted with respect to the threshold of 0.5 in a time series provided to us by S. Younger.

Authorized licensed use limited to: Akira Imada. Downloaded on May 16, 2009 at 19:53 from IEEE Xplore. Restrictions apply.

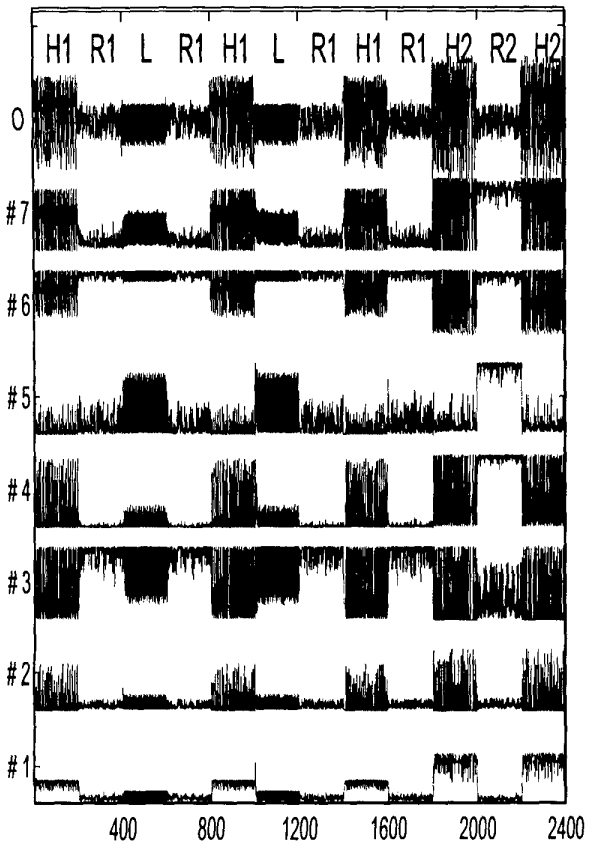

Figure 2: Outputs of nodes of the first hidden layer of the RMLP of [7]. The panel represents 12 different segments of the time series for five different types of behavior. These are denoted as follows: H1 and H2 stand for Henon map, types 1 and 2, respectively; L is a scaled logistic map; R1 and R2 are random outputs of two types. The uppermost plot illustrates the network's output. The horizontal grid lines are separated by 2.5. The outputs of all seven nodes are denoted as # with the node index. Though their values are in the range $[-1.0, +1.0]$, their plots are shifted appropriately for better visibility.

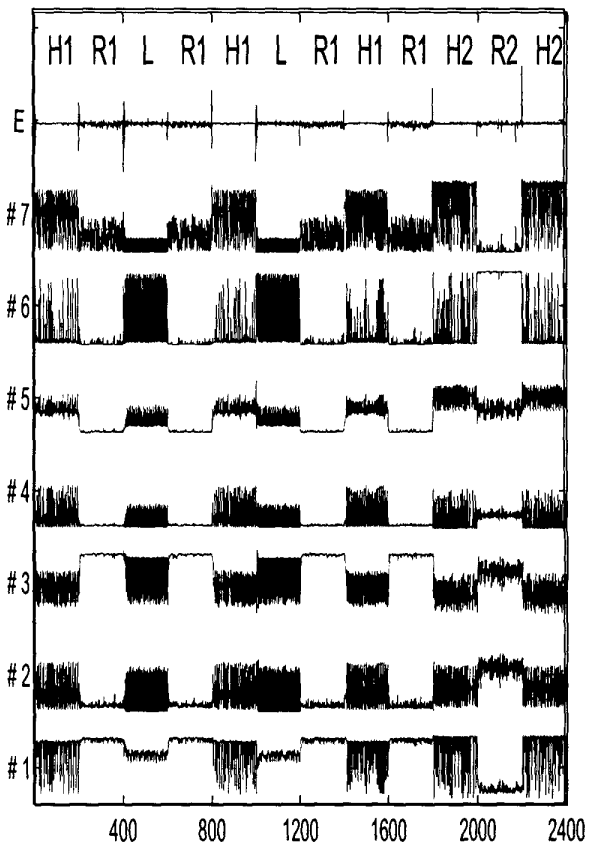

Figure 3: Outputs of nodes of the second hidden layer of the RMLP of [7]. The uppermost plot illustrates the network's error $y^d(t) - \hat{y}(t)$. The rest of the notation is the same as in the previous figure.

Authorized licensed use limited to: Akira Imada. Downloaded on May 16, 2009 at 19:53 from IEEE Xplore. Restrictions apply.

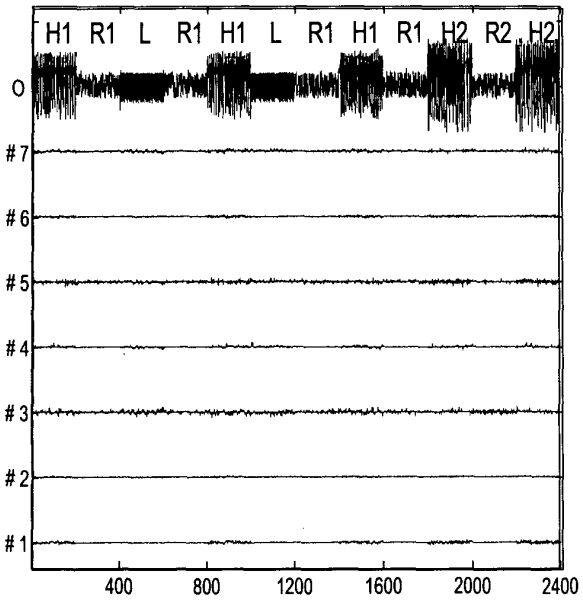

Figure 4: Variations of the outputs of nodes of the first hidden layer of the RMLP of [7] when the input is corrupted by the uniform noise. The notation is the same as in Figure 2.

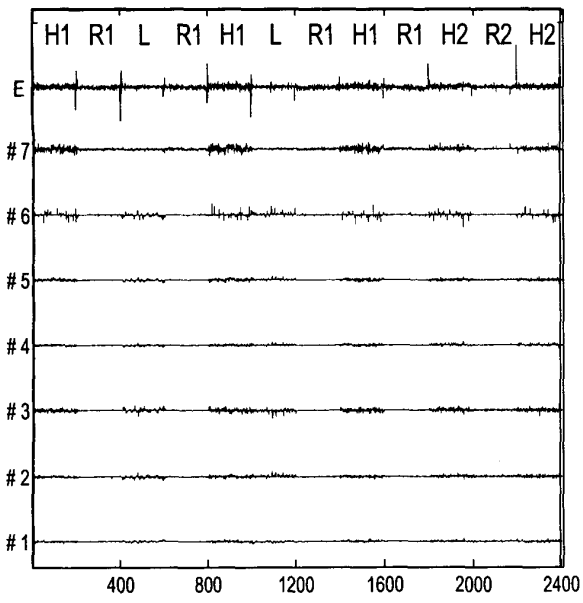

Figure 5: Variations of the outputs of nodes of the second hidden layer of the RMLP of [7] when the input is corrupted by the uniform noise. The notation is the same as in Figure 3. Note the slightly larger values of the output error, as compared to those in Figure 3.

eventually lose its grip on a small-error-level solution and fail.

3. What is the behavioral capacity of RNN? That is, can a greater number of meaningful mappings be "squeezed" into RNN of the fixed size? Experiments suggest that sometimes the capacity is very large, but othertimes it is not (e.g., in [7][2]). In any event, it is reasonable to ask whether many behaviors can be always induced reliably via training. While we are aware of recent results in [11] on capacity of RNN approximating discrete finite automata, it remains to be seen if these can be applied to meta-learning tasks discussed here.

These issues need to be addressed by both practitioners and theorists in future work.

References

[1] L. Feldkamp, G. Puskorius, and P. Moore, "Adaptation from Fixed Weight Dynamic Networks," in *Proc. of the IEEE International Conference on Neural Networks*, 1996.

[2] L. Feldkamp and G. Puskorius, "Fixed-Weight Controller for Multiple Systems," in *Proc. of the International Joint Conference on Neural Networks*, pp. 2268-2272, 1997.

[3] S. Younger, P. Conwell, and N. Cotter, "Fixed-Weight On-Line Learning," *Trans. on Neural Networks*, Vol.10, No.2, pp. 272-283, 1999.

[4] J. Lo, "Adaptive vs. Accommodative Neural Networks for Adaptive System Identification," in *Proc. of the International Joint Conference on Neural Networks*, pp. 1279-1284, 2001.

[5] S. Younger, S. Hochreiter, and P. Conwell, "Meta-Learning with Backpropagation," in Proc. of the International Joint Conference on Neural Networks, pp. 2001-2006, 2001.

[6] S. Hochreiter, S. Younger, and P. Conwell, "Learning to Learn Using Gradient Descent," in *Proc. of ICANN*, pp. 87-94, 2001.

[7] L. Feldkamp, D. Prokhorov, and T. Feldkamp, "Conditioned Adaptive Behavior from a Fixed Neural Network," in *Proc. of the 11th Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, pp. 78-83, 2001.

[8] D. Prokhorov, G. Puskorius, and L. Feldkamp, "Dynamical Neural Networks for Control," see in [11].

[9] L. Feldkamp and G. Puskorius, "A Signal Processing Framework Based on Dynamic Neural Networks with Application to Problems in Adaptation, Filtering, and Classification," *Proc. of IEEE*, Vol.86, No.11, pp. 2259-2277, 1998.

[10] A. Back and T. Chen, "Approximation of Hybrid Systems by Neural Networks," in *Proc. of ICONIP*, 1997.

[11] *A Field Guide to Dynamical Recurrent Networks*, J. Kolen and S. Kremer (Eds.), IEEE Press, 2001.

[2] It was noted that a smaller RMLP with 10 states (1-5R-5R-1L) did not appear likely to be trainable to yield a satisfactory solution, but an RMLP with 14 states did.

Authorized licensed use limited to: Akira Imada. Downloaded on May 16, 2009 at 19:53 from IEEE Xplore. Restrictions apply.