1 Backpropagation Through Time and Derivative Adaptive Critics: A Common Framework for Comparison[†]

Danil V. Prokhorov

Research and Advanced Engineering, Ford Motor Company, 2101 Village Rd., MD 2036, Dearborn, MI 48124, U.S.A., dprokhor@ford.com

1.1 INTRODUCTION

Heterogeneous ordered networks or, more specifically, recurrent neural networks (RNN) are convenient and flexible computational structures applicable to a broad spectrum of problems in system modeling and control. They require efficient algorithms to achieve successful learning for the given task. Backpropagation through time (BPTT), henceforth assumed to be its truncated version with a sufficient depth of truncation, and derivative adaptive critics (DAC) are two seemingly quite different approaches to solve temporal differentiable optimization problems with continuous variables. In fact, both BPTT and DAC are means to obtain derivatives for training parameters of RNN.

We show that both approaches are related. BPTT is used in DAC to obtain targets for derivative critic adaptation in RNN training. DAC can be interpreted as a method to reduce the need for introducing a potentially large truncation depth in BPTT by providing estimates of derivatives from the future time steps. This realization allows us to establish a common framework for comparison of derivatives of BPTT and those of DAC and summarize their differences. The main difference stems from the fact that derivatives provided by DAC are learned via a representation (critic network), and such derivatives can be averages of derivatives provided by BPTT. It should be kept in mind that some derivative averaging naturally occurs in the training process during which RNN parameters are being adjusted (usually incrementally).

[†]Portions of this chapter were previously published in [7], [9], [12], [13], [14] and [23].

Depending on the problem setup and critic training mechanics, DAC derivatives may or may not be sufficiently accurate for successful RNN training.

Both BPTT and DAC must also be equipped with a parameter adjustment rule or algorithm. BPTT equipped with various forms of the Kalman filter algorithm has shown its power in tackling difficult RNN training problems. Training a single RNN to model or control multiple systems with its weights fixed upon completion of the training is particularly remarkable because it defies successful applications of feedforward or time-delay NN. In spite of several successful demonstrations of DAC and in contrast to BPTT, DAC has mostly been restricted to training feedforward NN (neurocontrollers) using a gradient descent rule. It is crucial for DAC to reinforce itself with more powerful architectures and training algorithms to be capable of solving truly difficult optimization problems.

Sufficiently detailed comparisons between BPTT and DAC as training approaches to RNN are essentially lacking. Careful comparisons should be based not only on the results of comprehensive testing of the solutions but also on assessments of the computational requirements of the approaches. It is noteworthy that the critic network is discarded as soon as RNN parameter training is finished, which is wasteful. Furthermore, comparisons for a clearly formulated and easily accessible modeling problem may be preferable over comparisons for control problems because modeling problems usually have a relatively simple setup. We suggest a nonstationary system modeling problem as a possible benchmark for comparing BPTT and DAC.

This chapter is structured as follows. In Section 1.2 we show a relationship between BPTT and DAC that gives rise to a common framework for comparison of the two methods. In Section 1.3 we discuss critic representation. In Section 1.4 we propose a hybrid between BPTT and DAC which can be useful for comparative studies. In Section 1.5 we emphasize the need to base comparisons of the two methods not only on the final result but also on computational requirements for each method. We discuss two classes of challenging problems which could form a core of future comparative studies of BPTT and DAC in Section 1.6.

1.2 RELATIONSHIP BETWEEN BPTT AND DAC

We would like to show how BPTT is used within the DAC approach. We consider differentiable optimization with criterion

$$J(k) \quad = \quad \frac{1}{2}\sum_{t=k}^{k+h}\gamma^{t-k}\sum_{j=N_1}^{N_2}U_j^2(t) \tag{1.1}$$

where $0 < \gamma \leq 1$ is a discount, depth (horizon) h is as large as required, $U_j(t)$ is an instantaneous cost (or utility) function. Without loss of generality, each U_j is assumed to be a function of state variables of the following *ordered* heterogeneous network

$$x_i(t) = x_i^{ext}(t), \quad 1 \le i \le m \tag{1.2}$$

$$net_i(t) = \sum_{j=1}^{i-1} W_{i,j}(t)x_j(t) + \sum_{j=m+1}^{N} W_{i,j}^1 x_j(t-1) \tag{1.3}$$

$$x_i(t) = f_i(net_i(t)), \quad m+1 \le i \le N \tag{1.4}$$

where $x_i(\cdot) \in R$, $f_i(\cdot) \in C^1$, m is a number of external inputs x_i^{ext} to the network, and $m+1 \le (N_1, N_2) \le N$ is a set of indexes for which U_j are defined. The execution order is assumed to be from node 1 to node N. We want to determine parameters $W_{i,j}, W_{i,j}^1$ delivering a minimum to (1.1) in the mean square sense in the domain of interest $X : x_i \in X$.

Ordered derivatives [1] of the criterion J with respect to x_i are determined by taking into account (1.2)–(1.4):

$$\text{F\_}x_i(t) = E + F + R \tag{1.5}$$

where

$$E = U_i(t)\frac{\partial U_i(t)}{\partial x_i(t)} \tag{1.6}$$

$$F = \sum_{j=i+1}^{N} W_{j,i}(t)\frac{\partial f_j(net_j(t))}{\partial net_j(t)}\text{F\_}x_j(t) \tag{1.7}$$

$$R = \gamma \sum_{j=m+1}^{N} W_{j,i}^1(t+1)\frac{\partial f_j(net_j(t+1))}{\partial net_j(t+1)}\text{F\_}x_j(t+1) \tag{1.8}$$

Equation (1.5) is run backwards in *both* space ($i = N, N-1, ..., 1$) and time ($t = k+h, k+h-1, ..., k$) initializing $\text{F\_}x_i(k+h+1) = 0$ and $W_{j,i}^1(k+h+1) = W_{j,i}^1(k+h)$.

The expression for $\text{F\_}x_i(t)$ consists of three components. The term E of (1.6) is an explicit derivative of $1/2U_i^2(t)$ with respect to $x_i(t)$ (if exists). If the node x_i feeds other nodes through feedforward connections, then F of (1.7) should reflect all such connections. Likewise, R of (1.8) reflects all time-delayed connections through which the node x_i feeds others.

Ordered derivatives with respect to parameters $\mathbf{W}$ and $\mathbf{W}^1$ are determined using $\text{F\_}x_i(t)$ and (1.2)–(1.4):

$$\text{F\_}W_{i,j}(t) = \text{F\_}x_i(t)\frac{\partial f_i(net_i(t))}{\partial net_i(t)}x_j(t) \tag{1.9}$$

$$\text{F\_}W_{i,j}^1(t) = \text{F\_}x_i(t)\frac{\partial f_i(net_i(t))}{\partial net_i(t)}x_j(t-1) \tag{1.10}$$

Equations (1.5)–(1.10) are called BPTT [1]. Here they express a truncated form of BPTT, henceforth denoted as BPTT(h) [2].

Updates of parameters $\mathbf{W}$ and $\mathbf{W}^1$ can be made by using $\sum_{t=t_0}^{t_f} \text{F\_}W_{i,j}(t)$ and $\sum_{t=t_0}^{t_f} \text{F\_}W_{i,j}^1(t)$, respectively, where t_0, t_f are suitably chosen time steps of the trajectory (e.g., $t_0 = k$, $t_f = k + h$, another possibility is $t_0 = t_f = k$).

We now consider another optimization criterion (cf. (1.1)):

$$J'(t) \quad = \quad \frac{1}{2} \sum_{j=N_1}^{N_2} U_j^2(t) + \gamma J'(t+1) \tag{1.11}$$

The criterion J of (1.1) is an approximation of J' which becomes increasingly more accurate for sufficiently large h and $\gamma < 1$. On the other hand, the criterion (1.11) is approximated by J critic in approximate (heuristic) dynamic programming [3].

Let us examine derivatives of $J'(t)$ with respect to $x_i(t)$ taking into account (1.2)–(1.4):

$$\lambda_i'(t) \quad = \quad E' + F' + R' \tag{1.12}$$

where

$$E' \quad = \quad U_i(t) \frac{\partial U_i(t)}{\partial x_i(t)} \tag{1.13}$$

$$F' \quad = \quad \sum_{j=i+1}^{N} W_{j,i}(t) \frac{\partial f_j(net_j(t))}{\partial net_j(t)} \lambda_j'(t) \tag{1.14}$$

$$R' \quad = \quad \gamma \sum_{j=m+1}^{N} W_{j,i}^1(t+1) \frac{\partial f_j(net_j(t+1))}{\partial net_j(t+1)} \lambda_j'(t+1) \tag{1.15}$$

where $\lambda_j' \equiv \partial J'/\partial x_i$. Note that equation (1.12) is **nearly identical to (1.5) expressing backpropagation between two consecutive moments in time (BPTT(1)) [4]**. (Replacing $\lambda_i'(t)$ and $\lambda_j'(t+1)$ in (1.12),(1.14),(1.15) with $\text{F\_}x_i(t)$ and $\text{F\_}x_j(t+1)$, respectively, makes them identical.) Equations (1.9)–(1.10) may also be used for minimization of (1.11) except that $\text{F\_}x_i(t)$ should be replaced by $\lambda_i'(t)$.

In a popular DAC approach called *dual heuristic programming* (DHP), each λ_i' is to be approximated by output λ_i of λ critic [3]. One can demonstrate that traditional DHP equations (see, e.g., equations (7) and (8) in [5]) are a special case of (1.12). The λ critic is expressed as a suitable representation $\boldsymbol{\lambda}(\mathbf{x}(t), \mathbf{W}_C)$ with outputs $\lambda_i(\cdot)$ and adjustable weights $\mathbf{W}_C$. The critic is supposed to be trained with the error between $\lambda_i'(t)$ from (1.12) and its corresponding output $\lambda_i(t)$. (Each $\lambda_j'(t+1)$ in (1.15) is replaced by the appropriate output $\lambda_j(t+1)$ of the critic.)

We just revealed the similarity between a particular form of BPTT and a popular DAC formulation for the network (1.2)–(1.4), but this is also valid for much more general networks (see [4] and Section 1.4 of this chapter) and systems including those with distributed parameters [6]. Recognizing this similarity enables us to create a hybrid of BPTT and DAC in which DAC may act as a means to reduce depth h in

BPTT(h) by providing estimates of derivatives from the future time steps (see Section 1.4).

Many researchers have pointed out similarities between BPTT and the Euler-Lagrange/Hamiltonian formulation (see, e.g., [19], [17]). The expression (1.12) or (1.5) may be recognized as the Lagrange multipliers (partial derivatives of a Hamiltonian with respect to state variables) for the network (1.2)–(1.4) with the criterion (1.11) [18], [17]. The derivatives (1.9), (1.10), or partial derivatives of a Hamiltonian with respect to controls $\mathbf{W}$ and $\mathbf{W}^1$, can be used to update incrementally the network parameters in order to minimize the criterion (1.11).

We describe a typical application of BPTT to functional minimization. First, we initialize the state variables of (1.2)–(1.4). We run (1.2)–(1.4) forward for one or more time steps and compute the appropriate values of $U(t) : U(t), U(t+1), ..., U(t+k)$, where U is a known function of state variables and their targets (e.g., U is a tracking error). We then compute the derivatives according to (1.5) by backpropagating from $t+k$ to t (BPTT(k)) and perform the incremental updates based on (1.9) and (1.10).

Compared to this description of BPTT application, the DAC approach requires the utilization of only two adjacent time steps, t and $t+1$, according to (1.12). We run (1.2)–(1.4) forward for one time step, obtain $U(t)$, and invoke (1.12) to prepare for critic training. The right-hand side of (1.12) serves as both the set of instantaneous targets for critic training and the input to the parameter updates (1.9) and (1.10) (in place of F_$x_i(t)$). If gradient descent is employed, critic training is also incremental, and it may be based on the expression $(\lambda_i'(t) - \lambda_i(t))\frac{\partial \lambda_i(t)}{\mathbf{W}_C}$, where $\mathbf{W}_C$ is a vector of critic weights.

For either the BPTT or the DAC approach, we continue the training process for the next point $t+1$ along the trajectory. We can train for a segment of the trajectory, then reinitialize the state variables (1.2)–(1.4) and move on to training on another segment of the trajectory. Meanwhile, our weights $\mathbf{W}$, $\mathbf{W}^1$ (and $\mathbf{W}_C$ for DAC) serve as the long-term memory, incorporating the effects of adaptation averaged over many instances. For an adequately chosen training strategy, we can reasonably expect that application of BPTT or DAC will result in the triplet $(\mathbf{W}, \mathbf{W}^1, \mathbf{W}_C)^*$ such that J is approximately minimized over X. As with any numerical and (generally) nonconvex optimization, all we can guarantee is that the proper training process should result in attaining a local minimum of J (most of the time such a minimum is satisfactory). What is proper remains problem dependent, but determining the training strategy includes choosing training parameters (e.g., learning rates) for updates based on (1.9) and (1.10), the length and the assignment of trajectory segments and the initialization of state variables. For DAC we need to add the choice of critic training parameters and the coordination scheme between critic and network training processes [15], [4].

The training process based on BPTT also resembles a form of model predictive control (MPC) with receding horizon (see, e.g., [20]). As in the MPC, we run the system forward for several time steps collecting values of U. Our horizon ($t+k$) recedes once the weight updates are carried out except that our updates are incremental, not “greedy" as in the receding-horizon MPC.

We summarize the differences between derivatives obtained by BPTT and those of DAC [7]:

1) BPTT derivatives are computed directly, while critic derivatives are computed from a representation, e.g., a neural network, with its parameters to be learned.
2) BPTT(h) derivatives generally involve a finite time horizon (equal to the chosen *truncation depth* h), while critic derivatives are estimates for an infinite horizon; it is not uncommon, however, to employ a discount factor in (1.11), which may be interpreted as a gentle truncation ($\gamma < 1$). Large h are permissible because BPTT computations scale linearly with h, but frequently small h ($h < 10$) suffice.
3) BPTT derivatives necessarily compute the effect of changing a variable in the *past*, while a derivative critic may be used to estimate the effect of a change at the present time. If critics are used only to adjust network parameters, this distinction is **irrelevant**. On the other hand, recognizing it poses interesting possibilities for alternative or supplementary methods of control, as discussed in Section 6 of [7].
4) A BPTT derivative is essentially exact for the specific trajectory for which it is computed, while a critic derivative necessarily represents an average of some kind, e.g., an average over trajectories that begin with a given system state. Such an estimate may be quite accurate (if the critic has been well adapted or trained and if exogenous inputs to the system are either small or statistically well behaved) or may be essentially worthless (if future operation is completely unpredictable due to arbitrary inputs or targets).

Item 4 is discussed in more details in the next section.

1.3 CRITIC REPRESENTATION

Critic predicts the effect of a change in a variable of the system or network (1.2)–(1.4) on its future operation. Critic is thus a function of the system state, and it is important to include in critic representation as much information as available about the system. What is encompassed by the system depends on the context. In the context of indirect model reference adaptive control [8], the system is interpreted quite generally as consisting of: 1) controller network, 2) reference model, 3) object to be controlled (plant), and 4) model of the plant. State variables or their estimates of *all* of these components should be provided as inputs to the critic (the plant model often serves as the plant state estimator). However, the only adjustable quantities are the weights of controller and (sometimes) parameters of the model.

It is convenient to consider all main components of the control system as parts of a *single* heterogeneous recurrent network, perhaps similar to (1.2)–(1.4). This viewpoint equates control problems with modeling problems (employing RNN) since both types of problems feature feedback.

In the modeling context, we suggest connecting all recurrent nodes or time-delayed elements to the critic because they reflect the *state* of the system. In the example below we illustrate that adding a state variable is indeed beneficial [4].

Consider a system

$$x(k+1) = x^{ext}(k+1) + wx(k) \tag{1.16}$$

$$x^d(k+1) = x^{ext}(k+1) + w^d x(k) \tag{1.17}$$

where $0 < |w| < 1$, $0 < |w^d| < 1$, and $x^{ext}(k), \forall k$ are i.i.d. random variables with the same mean x^{ext}. The system (1.16)-(1.17) is a simple illustration of the network (1.2)–(1.4), with the controller part expressed as (1.16). The equation (1.17) is an example of the reference model, and it describes the desired behavior of (1.16).

The minimization criterion is

$$J(k) = \frac{1}{2}\sum_{t=0}^{\infty} \gamma^t U^2(k+t) = \frac{1}{2}\sum_{t=0}^{\infty} \gamma^t (x^d(k+t) - x(k+t))^2 \tag{1.18}$$

Differentiating $J(k)$ with respect to $x(k)$ yields

$$\text{F\_}x(k) \quad = \quad \sum_{t=0}^{\infty} (\gamma w)^t (x(k+t) - x^d(k+t)) \tag{1.19}$$

Substituting (1.16) and (1.17) into the equation for $\text{F\_}x(k)$ above, averaging for all $x^{ext}(\cdot)$ and recognizing appropriate power series results in

$$\begin{aligned} < \text{F\_}x(k) > \quad &= \quad \frac{x(k)}{1-\gamma w^2} - \frac{x^d(k)}{1-\gamma w w^d} \\ &\quad + \frac{x^{ext}\gamma w}{1-\gamma w}\left(\frac{1}{1-\gamma w^2} - \frac{1}{1-\gamma w w^d}\right) \end{aligned} \tag{1.20}$$

where $< \text{F\_}x(k) >$ is an (ensemble) average ordered derivative of $J(k)$ with respect to $x(k)$.

We wish to show that a linear λ critic is sufficient to recover $< \text{F\_}x(k) >$. The critic representation is

$$\lambda(t) \quad = \quad Ax(t) + Bx^d(t) + C \tag{1.21}$$

where A, B, and C are the critic weights. Similar to (1.12), we can write

$$\lambda(t) \quad = \quad E' + R' \tag{1.22}$$

where

$$E' \quad = \quad x(t) - x^d(t) \tag{1.23}$$

$$R' \quad = \quad \gamma\lambda(t+1)\frac{\partial x(t+1)}{x(t)} \tag{1.24}$$

Here (1.23) corresponds to (1.13), and (1.24) corresponds to (1.15) (there is no feedforward part like (1.14)). Substituting the representation (1.21) into the equation for $\lambda(t)$ above yields

$$\begin{aligned} Ax(t) + Bx^d(t) + C \quad &= \quad x(t) - x^d(t) \\ &\quad + \gamma w(Ax(t+1) + Bx^d(t+1) + C) \end{aligned} \tag{1.25}$$

When the weights converge they become

$$A = \frac{1}{1-\gamma w^2} \tag{1.26}$$
$$B = -\frac{1}{1-\gamma w w^d} \tag{1.27}$$
$$C = \frac{x^{ext}\gamma w}{1-\gamma w}\left(\frac{1}{1-\gamma w^2} - \frac{1}{1-\gamma w w^d}\right) \tag{1.28}$$

which, incidentally, indicates that $<$ F_$x(k) >$ is restored exactly.

A reasonable question to ask is what happens if we do not have access to state(s) of the data generator/reference model x^d. Such situation may happen when training a recurrent network from a file of input-output pairs. We suggest approximating the states of the missing reference model by its time-delayed outputs and provide these estimates as inputs to the critic. (The order of the system which produced the training data might be known and used to determine how many time-delayed inputs to employ.)

One can certainly argue that even a simpler critic representation lacking some difficult-to-find inputs might suffice for a problem at hand. However, we should keep in mind that excluding a state or its estimate from the critic input set effectively turns such a state into a disturbance which tends to decrease the likelihood of getting an accurate critic and handicaps the critic as compared to BPTT.

Many representations for critics are possible. For example, each output can be a separate function or neural network $\lambda_i(\mathbf{x}(t), \mathbf{W}_C)$. In the example above we used the linear critic. Extremely simple (bias weight only) representations are also viable for some problems [9], as illustrated below.

Consider the following system

$$x_1(t) = bx^{ext}(t) \tag{1.29}$$
$$x_2(t+1) = x_1(t) \tag{1.30}$$
$$x_3(t+1) = 0.5x_3(t) + x_1(t+1) + x^{ext}(t+1) - 2x_2(t) \tag{1.31}$$

or, in a compact form,

$$x_3(t+1) = 0.5x_3(t) + (1+b)x^{ext}(t+1) - 2bx^{ext}(t-1) \tag{1.32}$$

The reference input x^{ext} has a piecewise constant pattern with a long enough dwell time (e.g., 50 time steps). The goal is to adjust the parameter b so as to minimize $(x_3^d(t) - x_3(t))^2$, where $x_3^d(t) \equiv 0, \forall t$, in the mean square sense in which case $b = 1$. BPTT(2) must be used to accomplish this (the training process diverges if BPTT(h) with $h < 2$ is used).

We show how the use of two DAC (λ) critics eliminates the need for more than the (computational) equivalent of BPTT(1). The BPTT equations for this system are similar to (1.5):

$$\begin{aligned} \mathrm{F\_}x_3(t) &= -e(t) + 0.5\mathrm{F\_}x_3(t+1) && (1.33)\\ \mathrm{F\_}x_2(t) &= -2\mathrm{F\_}x_3(t+1) && (1.34)\\ \mathrm{F\_}x_1(t) &= \mathrm{F\_}x_3(t) + \mathrm{F\_}x_2(t+1) && (1.35) \end{aligned}$$

where $e(t) = 0 - x_3(t) = -x_3(t)$. In the first equation $-e(t)$ is equivalent to (1.6), and $0.5\mathrm{F\_}x_3(t+1)$ is equivalent to (1.8) (it is obtained from the equation for $x_3(t+1)$). The equation for $x_3(t+1)$ is also used to obtain $\mathrm{F\_}x_2(t)$ and $\mathrm{F\_}x_3(t)$ in the expression for $\mathrm{F\_}x_1(t)$. The last equation (for $\mathrm{F\_}x_1(t)$) is obtained by combining $\mathrm{F\_}x_1(t) = \mathrm{F\_}x_2(t+1)$ with $\mathrm{F\_}x_1(t+1) = \mathrm{F\_}x_3(t+1)$ at time t.

The BPTT equations above are to be repeated two times (BPTT(2)) to produce $\mathrm{F\_}x_1$ suitable for updating b correctly:

$$\mathrm{F\_}b(t) = \mathrm{F\_}x_1(t)x^{ext}(t) \qquad (1.36)$$

According to (1.12), we replace $\mathrm{F\_}x_3(t+1)$ and $\mathrm{F\_}x_2(t+1)$ with $\lambda_3(t+1)$ and $\lambda_2(t+1)$, respectively, and obtain

$$\begin{aligned} \lambda_3'(t) &= -e(t) + 0.5\lambda_3(t+1) && (1.37)\\ \lambda_2'(t) &= -2\lambda_3(t+1) && (1.38)\\ \mathrm{F\_}x_1(t) &= \lambda_3'(t) + \lambda_2(t+1) && (1.39) \end{aligned}$$

where $\mathrm{F\_}x_1(t)$ is used in (1.36) to update b. It turns out that the bias-weight-only critics suffice for this problem, and they may be updated as follows

$$\begin{aligned} \lambda_3 &\mathrel{+}= \eta(\lambda_3'(t) - \lambda_3) && (1.40)\\ \lambda_2 &\mathrel{+}= \eta(\lambda_2'(t) - \lambda_2) && (1.41) \end{aligned}$$

where learning rate $\eta > 0$ is reasonably chosen, and the C-language notation "$+=$" indicates that the quantity on the right hand side is to be added to the previously computed value of the left hand side. It can be shown that performing updates of the critics and the weight b results in convergence of b to its desired value of unity if the system is sufficiently excited by x^{ext}.

The system (1.32) could easily be changed to represent D delays which would either require BPTT(D) or D critics trained in a fashion similar to this example.

As illustrated with the example (1.29)–(1.41), even a trivial critic can be effective and competitive with BPTT when dealing with a predictable system. (A good example of a predictable system is a system with a constant (or fixed-statistics random) disturbance.) The system above is predictable because it is driven by a slowly-varying excitation. However, any critic-based training approach will have difficulties if the system is subject to x^{ext} changing significantly at every time step. For this and other systems with recurrence and block delays affected by fast-changing disturbances or excitations it is better to use a hybrid of temporal backpropagation [10], BPTT and, possibly, DAC proposed in [11].

1.4 HYBRID OF BPTT AND DAC

Here we provide the reader with C-language-style pseudocode describing a hybrid of BPTT and DAC which is less efficient in handling block delays than that in [11] but easier to implement.

The forward equations for an ordered network with `n_in` inputs and `n_out` outputs may be expressed very compactly in a pseudocode format [13]. Let the network consist of `n_nodes` nodes, of which `n_in` serve as receptors for the external inputs. The bias input, which we denote formally as node 0, is not included in the node count. The bias input is set to the constant 1.0. The array **I** contains a list of the input nodes; e.g., I_j is the number of the node that corresponds to the `j`th input, in_j. Similarly, a list of the nodes that correspond to network outputs out_p is contained in the array **O**. We allow network outputs and targets to be advanced or delayed with respect to node outputs by assigning a phase τ_p to each output. For example, if we wish to associate the network output p with the output of some system five steps in the future, we would have $\tau_p = 5$. Node i receives input from `n_con(i)` other nodes and has activation function $f_i(\cdot)$; `n_con(i)` is zero if node `i` is among the nodes listed in the input array **I**. The array **c** specifies connections between nodes; $\mathrm{c}_{i,j}$ is the node number for the `j`th input of node `i`. Inputs to a given node may originate at the current or past time steps, as specified by delays contained in the array **d**, and through weights for time step t contained in the array $\mathbf{W}(t)$.

Prior to beginning network operation, all appropriate memory is initialized. Normally, such memory will be set to zero. (In some cases memory that corresponds to the network initial state may be set to specified values.)

At the beginning of each time step, we execute the following buffer operations on weights and node outputs (in practical implementation, a circular buffer and pointer arithmetic may be employed). Here `dmax` is the largest value of delay represented in the array **d**, and `h` is the truncation depth of the BPTT gradient calculation described in a pseudocode form later.

```
for i = 1 to n_nodes {
 for i_t = t-h-dmax to t-1 {
```

$$\mathbf{W}(i_t) = \mathbf{W}(i_t + 1) \quad (1.42)$$

$$y_i(i_t) = y_i(i_t + 1) \quad (1.43)$$

```
 } /* end i_t loop */
} /* end i loop */
```

Then, the actual network execution is expressed as

```
for i = 1 to n_in {
```

$$y_{\mathrm{I_i}}(t) = \mathrm{in}_i(t) \quad (1.44)$$

```
}
for i = 1 to n_nodes {
 if n_con(i) > 0 {
```

$$a_i(t) = \left[\begin{array}{c} \sum_{j=1}^{\text{n\_con(i)}} W_{i,j}(t) y_{c_{i,j}}(t-d_{i,j}) \\ \prod_{j=1}^{\text{n\_con(i)}} (W_{i,j}(t) + y_{c_{i,j}}(t-d_{i,j})) \end{array} \right. \tag{1.45}$$

$$y_i(t) = f_i\big(a_i(t)\big) \tag{1.46}$$

```
 }
}
for p = 1 to n_out {
```

$$\text{out}_p(t+\tau_p) = y_{\text{O}_\text{p}}(t) \tag{1.47}$$

```
}
```

Most commonly, we take the (differentiable) activation function $f_i(\cdot)$ to be either linear or a bipolar sigmoid, though we also can make use of other functions, e.g., sinusoids, for special purposes. In the pseudocode above, the top portion of the right-hand side of (1.45) is invoked whenever the node i performs a summation of its inputs weighted by the appropriate elements of $\mathbf{W}$. The bottom portion of the right-hand side of (1.45) is invoked if the node i is a product (multiplicative) node.

The pseudocode above is very general, and it can be used to describe a great deal of neural and non-neural computational structures including (1.2)–(1.4).

In the pseudocode for a hybrid of *aggregate* BPTT [12] and DAC below, $\text{F\_}^p y$ denotes an ordered derivative of $\frac{1}{2}\sum_{i_\text{h}=0}^{h} \gamma^{i_\text{h}} \left(U_p(t-h+i_\text{h})\right)^2$ (cf. (1.1)), where U_p is a component of the utility vector $\mathbf{U}$ usually expressed as a deviation between appropriate target and output of the network. This pseudocode can be invoked at each time step only after the completion of forward propagation at time step t.

```
for p = 1 to n_out {
 for i = 1 to n_nodes {
  for k = 1 to n_con(i) {
```

$$\text{F\_}^p W_{i,k} = 0 \tag{1.48}$$

```
  } /* end k loop */
  for i_t = t to t-h-dmax {
```

$$\text{F\_}^p y_i(i_t) = 0 \tag{1.49}$$

```
   if i_t = t {
```

$$\mathrm{F\_}^{p}y_i(i_t) \quad = \quad \kappa_i^p(i_t) \tag{1.50}$$

```
   }
  } /* end i_t loop */
 } /* end i loop */
 for i_h = 0 to h {
```

$$\begin{aligned} i_1 &= \max(t - i_h, 0) && (1.51)\\ U_p(i_1) &= \mathrm{tgt}_p(i_1 + \tau_p) - \mathrm{out}_p(i_1 + \tau_p) && (1.52)\\ \mathrm{F\_}^{p}y_{\mathrm{O_p}}(i_1) &\mathrel{+}= -U_p(i_1) && (1.53) \end{aligned}$$

```
  for i = n_nodes to 1 {
   if n_con(i) > 0 {
    for k = n_con(i) to 1 {
```

$$\begin{aligned} j &= c_{i,k} && (1.54)\\ i_2 &= \max(i_1 - d_{i,k}, 0) && (1.55)\\ \mathrm{F\_}^{p}y_j(i_2) &\mathrel{+}= \gamma^{d_{i,k}}\mathrm{F\_}^{p}y_i(i_1) f_i'(a_i(i_1)) \\ &\quad \times \left[\frac{W_{i,k}(i_1)}{\prod_{m=1,m\neq k}^{\mathrm{n\_con(i)}}(W_{i,m}(i_1) + y_{c_{i,m}}(i_1 - d_{i,m}))} \right] && (1.56) \end{aligned}$$

$$\begin{aligned} \mathrm{F\_}^{p}W_{i,k} &\mathrel{+}= \mathrm{F\_}^{p}y_i(i_1) f_i'(a_i(i_1)) \\ &\quad \times \left[\frac{y_j(i_2)}{\prod_{m=1,m\neq k}^{\mathrm{n\_con(i)}}(W_{i,m}(i_1) + y_{c_{i,m}}(i_1 - d_{i,m}))} \right] && (1.57) \end{aligned}$$

```
    } /* end k loop */
   }
  } /* end i loop */
 } /* end i_h loop */
} /* end p loop */
```

The loops for (1.48) and (1.49) serve as initializations. We use κ_i^p to denote the output i of derivative critic for component p of the utility vector **U**. This is done to avoid confusion with λ_i discussed above.

By virtue of "$+=$" notation, the appropriate derivatives are distributed from a given node to all nodes and weights that feed it in the forward direction, with due allowance for any delays that might be present in each connection. The simplicity of the formulation reduces the need for visualizations such as unfolding in time or signal-flow graphs.

If the node i is one of the summation nodes (as in the top portion of the right-hand side of (1.45)), then the derivatives with respect to y_j and $W_{i,k}$ are computed according to the top portions of the left-bracketed terms in (1.56) and (1.57), respectively. For a multiplicative node (as in the bottom portion of the right-hand side of (1.45)), these derivatives are computed according to the bottom portions of the left-bracketed terms in (1.56) and (1.57).

DAC errors and targets are computed after completion of the pseudocode above:

```
for p = 1 to n_out {
 for i = n_nodes to 1 {
  if n_con(i) > 0 {
   for k = n_con(i) to 1 {
                      $j = c_{i,k}$                                          (1.58)
                      $i_2 = t - h - d_{i,k}$                                (1.59)
                      $\kappa_j^{*p}(i_2) = \mathrm{F}^p\_y_j(i_2)$          (1.60)
                      $\epsilon_j^p(i_2) = \kappa_j^{*p}(i_2) - \kappa_j^p(i_2)$   (1.61)
   } /* end k loop */
  }
 } /* end i loop */
} /* end p loop */
```

A special case with $h = 0$ and `dmax` $= 1$ is what may be called the pure DAC algorithm (cf. equation (1.12)). However, there is one crucial difference. In the usual formulation (1.12), derivative critics designated by λ are trained to estimate derivatives of J', including the contribution from the current step, k. The pseudocode above separates the estimate of the future from that of the present, and it is a generalization of the algorithm proposed in [14] (cf. C critics in [14]).

The difference between the values of the two critic forms is precisely the quantity that results when the derivative of error at each output node $\frac{\partial J'}{\partial y_{out}(k)}$ is backpropagated to $y(k)$. (In the simple case of a single-node network, the new critic $\kappa(k)$ is related to the usual critic as follows: $\lambda(k) = \frac{\partial J'}{\partial y(k)} + \kappa(k)$.) The κ critic is thus not required to estimate quantities which can be computed exactly. Limited experimentation suggests that the use of κ critics may lead to faster training than that of λ critics.

Critic can be trained using the error (1.61). For example, a gradient descent update of critic weights may look like this:

$$\mathbf{W}_C^p \quad += \quad \eta^p \epsilon_j^p(t-1) \frac{\partial \kappa_j^p(t-1)}{\partial \mathbf{W}_C^p} \tag{1.62}$$

where $\mathbf{W}_C^p$ is a vector of weights of the critic p, and $\eta^p > 0$ ($h = 0$ and $\texttt{dmax} = 1$). (We assume in (1.62) that an individual critic is used to approximate ordered derivatives of the discounted sum of U_p^2 with respect to node output y_j. One can also combine all such critics into one network.) The critic outputs may correspond to different time steps when $d_{i,k}$ is different for different RNN nodes, as follows from (1.61).

No critic training happens if all critic outputs κ_j^p and their learning rates are fixed at zero. In such a case, our pseudocode amounts to carrying out the aggregate BPTT(h). If the gradient descent training of weights $\mathbf{W}$ is desired, then we use F_$W_{i,k}^p$ of (1.57) to adjust $W_{i,k}(t)$. For extended Kalman filter (EKF) training [13], the error injection (1.53) should be modified for consistency with mechanics of the EKF recursion (see [12]).

If $h > 0$ and critic training is enabled, then (1.62) may be invoked to train the critic (alternatively, the EKF algorithm can be used). While limited experiments with utilizing derivatives (1.57) in conjunction with EKF updates have been carried out successfully, further experimentation is needed since such derivatives are different from those usually employed by the EKF, especially when backpropagation to the RNN weights, as in (1.57), is performed only for $i_1 = t - h$ [14].

Concluding this section, it should be mentioned that our hybrid can be used even when components of $\mathbf{U}$ are not defined for all time steps. Furthermore, a differentiable approximation of $\mathbf{U}$ may be used if the true $\mathbf{U}$ is not well defined. Such an approximation capturing an essential relationship between the network variables and the desired instantaneous utility or the final outcome (e.g., in a game setting) could be learned in a separate training task prior to invoking the hybrid equations of this section [7].

1.5 COMPUTATIONAL COMPLEXITY AND OTHER ISSUES

We wish to compare the overhead associated with computations of F_$W_{i,j}$ and κ_i for DAC with the cost of computing F_$W_{i,j}$ for BPTT(h).

We assume that the cost of forward and back propagations through a network is dominated by a linear term proportional to the number of its weights. For a critic with $N_{\mathbf{W}_C}$ weights, the cost of carrying out static BP (BPTT(0)) is $O(N_{\mathbf{W}_C})$. If we use total N_C data points to train a critic, the critic training cost is proportional to both $N_{\mathbf{W}_C}$ and $N_{\mathbf{W}}$ because backpropagation through a RNN with $N_{\mathbf{W}}$ weights to obtain critic targets (1.61) has $O(N_{\mathbf{W}})$ complexity for each of N_C data points. Training a network with DAC on N_A data points incurs a cost proportional to both $N_{\mathbf{W}}$ and $N_{\mathbf{W}_C}$ because the critic is to be executed with $O(N_{\mathbf{W}_C})$ complexity for each of N_A data points. Thus, the total computational cost of DAC algorithm is proportional to $(N_C + N_A)N_{\mathbf{W}_C}$ and $(N_C + N_A)N_{\mathbf{W}}$.

The cost of BPTT(h) is $O(N_{\mathbf{W}}h)$. If we use N_B data points to train the network, then the total cost is proportional to $N_B N_{\mathbf{W}} h$.

Our simple analysis does not take into account the cost of updating $\mathbf{W}$ in both the network and the critic which can be significant, especially for second order methods.

Let us assume that the network weight updating incurs the same cost for both methods. If

$$\alpha(N_C + N_A)N_{\mathbf{W}_C} + \beta(N_C + N_A)N_{\mathbf{W}} + \mathbf{W}_C \text{ update cost} \quad < \quad \omega N_B N_{\mathbf{W}} h \tag{1.63}$$

where α, β, ω are some problem-dependent constants, then the DAC approach is more efficient computationally than the BPTT(h) approach **provided** that the updates of $\mathbf{W}$ result in the same network performance upon completion of its training.

It appears possible to simplify the DAC approach for some problems, e.g., when the system is affine in controls $\mathbf{a}$:

$$\mathbf{x}(t+1) \quad = \quad \mathbf{F}(\mathbf{x}(t)) + \mathbf{G}(\mathbf{x}(t))\mathbf{a}(t) \tag{1.64}$$

where $\mathbf{F}$ is a vector function, $\mathbf{G}$ is a control matrix. If $U(t)$ is quadratic in $\mathbf{a}$, i.e., includes the term $\mathbf{a}^T(t)\mathbf{R}\mathbf{a}(t)$, $\mathbf{R} > 0$, then implementable parameterization of the optimal controller may be expressed using the critic $\boldsymbol{\lambda}$ (vector) as the following product

$$\mathbf{a}(t) \quad = \quad -\mathbf{R}^{-1}\mathbf{G}^T(\mathbf{x}(t))\boldsymbol{\lambda}(\mathbf{x}(t), \mathbf{W}_C) \tag{1.65}$$

Thus unlike the usual DAC approach featuring training of two entities (critic and controller), no controller training is necessary here. In such a case the expression for computational cost comparison above is changed to

$$\zeta N_C N_{\mathbf{W}_C} + \mathbf{W}_C \text{ update cost} \quad < \quad \omega N_B N_{\mathbf{W}} h \tag{1.66}$$

where ζ is another constant.

Derivatives from DAC (values of λ_i or κ_i) represent averages of derivatives F_x_i, as shown in the example (1.16)–(1.28). That example also touches upon the following important issue. To obtain the accurate average $<$ F_$x(k)$ $>$, we have to wait until convergence of critic weights A,B and C. Even more importantly, the parameters w and w^d must be kept fixed. If training of w is in progress (or w^d is changing), critic accuracy will also depend on how these parameters are being changed, because every change in w or w^d results in a change to the system or network to which the critic is being adapted. In a general case of many weights changing in a network, if their updates are to be closely interleaved with critic updates, a relatively simple critic representational structure might be warranted, so that the critic can be quickly adapted to changes in the system.

The proper coordination of critic training with network (controller) training has been a research topic. We have found that, in some cases, it is possible to update both network weights and critic weights at every training step, although such a strategy may not work well in the presence of network weight updates of larger size, as frequently result from second order training procedures. Alternating the training processes in blocks is a reasonable option, since holding the network fixed while adapting the critic generally leads to greater critic accuracy. The drawback is that once the critic is held fixed and the network is changed, the critic may become very inaccurate and compromise training with poor derivatives. Hence a better approach might be to carry out the training processes concurrently but to monitor the critic error (the error used to update the critic) and to suspend network training for some number of steps if a specified critic error is exceeded. Of special interest are case studies in [15], [16] where various alternatives for coordinated updates of critic and network have been analyzed. One alternative includes training of more than one critic (termed “shadow critic/controller" method), with periodic alternations between the critics to improve convergence. Yet, no comparison from the standpoint of computational cost akin to the analysis of this section have been made.

Efficiency of critic adaptation is paramount because the critic is discarded as soon as the controller training is finished. Interestingly, very few attempts have been made to analyze critic training accuracy or critic usability after obtaining the required controller. As for the latter, we discuss the use of a critic to analyze Lyapunov stability of the closed-loop system in [21].

1.6 TWO CLASSES OF CHALLENGING PROBLEMS

BPTT(h) equipped with EKF algorithm [13] has shown its power in dealing with difficult training problems requiring the use of RNN. Recently progress has been made in nonlinear Kalman filters in a joint estimation framework with promise to eliminate not only BPTT but also the necessity to calculate derivatives in the system. Currently the new method’s only drawback is extra complexity as compared to that of the standard EKF method [22].

More challenging of RNN training problems solved with Kalman filter methods can be categorized in two broad classes [23]. Class I encompasses neural approximation of multiple input-output mappings of the following form

$$\mathbf{y}^d(t) = \mathbf{f}_\theta(\mathbf{z}_\theta(t-1), \mathbf{x}_\theta(t)) \tag{1.67}$$

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

$$\hat{\mathbf{y}}(t) = \mathbf{f}(\mathbf{z}(t-1), \mathbf{x}_\theta(t)) \tag{1.68}$$

where $\mathbf{z}$ is its state vector. Sometimes none of the mappings have states $\mathbf{z}_\theta$, as in [24] and [25]. 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:

$$\hat{\mathbf{y}}(t) = \mathbf{g}_\theta(\mathbf{z}_\theta(t-1), \mathbf{f}(\hat{\mathbf{y}}(t-1), \mathbf{z}(t-1), \mathbf{x}_\theta(t))) \tag{1.69}$$

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 [26]). 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.

We briefly describe examples of class-I problems. In [27], 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 [28], 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. This problem is impossible (or, at least, very inefficient) to solve with feedforward network equipped with a tapped-delay line because of the need to correctly maintain a potentially arbitrarily long response to the random input.

Two problems below are examples of class-II problems. In [26], a two-hidden-layer RMLP (14 states) is trained to act as a stabilizing controller for three distinct and unrelated systems, without explicit knowledge of system identity. This problem, too, has a feature which makes it very difficult (if not impossible) to apply successfully a controller based on a feedforward network equipped with a tapped-delay line. Specifically, the steady state values of controls for all three plants are quite different, yet the stabilization is required around the same equilibrium point (the origin).

In [29], 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. The robustness results of RMLP-based controller are shown to be much better than those of a controller based on a feedforward network.

These results obtained with BPTT(h) and EKF for clearly formulated and easily accessible control problems may serve as benchmarks for future comparison studies with DAC. Indeed, in spite of several successful demonstrations of DAC, they have mostly been limited to training controllers based on feedforward/time-delay neural networks using the gradient descent algorithm (see, e.g., [30], [31], [16], [32]), with the notable exception of [33].

Both class I and class II represent problems that are important and frequently observed in practice. For example, a physical system to be modeled or controlled is usually known only to within parametric or structural (possibly time-varying) uncertainties. (Such uncertainties amount to different, discrete or continuous sets of mappings $\mathbf{f}_\theta$ and $\mathbf{g}_\theta$.) One approach is to employ an adaptive system whose parameters would adapt in response to differences between the model and the reality. Another approach discussed in this section is to employ a RNN with *fixed* weights whose recurrent nodes would act as counterparts of parameters of the conventional adaptive system. This approach has an advantage of bypassing the thorny issue of adapting weights on-line.

In general, setup of class-II problems is more complex than that of class-I problems, therefore we suggest to try comparative studies initially on problems of class I. They share the same feature (feedback) with class-II problems due to the presence of recurrent connections in the network. We discuss one of class-I problems below.

1.6.1 Learning all quadratic functions

The problem of learning *all* quadratic functions of two variables is proposed in [24]. The quadratic functions are $y^d(t) = a(k){x_1}^2(t) + b(k){x_2}^2(t) + c(k)x_1(t)x_2(t) + d(k)x_1(t) + e(k)x_2(t) + f(k)$ where ranges for $a, b, c, d, e, f, x_1, x_2$ are the same: $[-1.0, 1.0]$. The index k (function index) changes discretely and much less frequently than the index t (example index). A special form of RNN called *long short-term memory* (LSTM) is explored in [24]. The LSTM has three inputs ($x_1(t)$, $x_2(t)$ and $y^d(t-1)$), one output ($\hat{y}(t)$), and it consists of 5,373 weights. Its training set is a time series of 128,000 points (128 different quadratic functions of 1000 examples each). The root mean square (RMS) error reaches 0.025 by the end of training. The final LSTM demonstrates the test RMS error of 0.026. It is claimed that other recurrent networks can not match performance of LSTM on this and other *metalearning* problems.

We can interpret the quadratic function problem as a modeling problem of a non-stationary time series. Indeed, only x_1,x_2 and y^d are observed, and $(a, b, c, d, e, f)_k$ forms a hidden state changing every so often. We want to train a RMLP in the same setting as that of the LSTM training experiment [25]. Our RMLP has the same three inputs, $x_1(t)$, $x_2(t)$ and $y^d(t-1)$, and architecture 3-30R-10R-1L with output $\hat{y}(t)$. (The notation 3-30R-10R-1L stands for RMLP with three inputs, 30 nodes in the first hidden fully recurrent layer, 10 nodes in the second hidden fully recurrent layer, and one linear output node.) It has 1441 weights. Values of y^d and $\hat{y}$ 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. The initial 40 points of each segment are used to let the network develop its states (*priming* operation) from their initial zero values, rather than for training weights. Next, we apply the 20-stream global EKF to update weights, with derivatives being computed by BPTT(40). We use 20×1000 points for training in each epoch. Our training session lasts for 1620 epochs, during which each data point was presented to the network approximately 250 times. The first 600 epochs are carried out with the parameter $R_0 = 100$ (measurement noise or inverse learning rate) and the parameter $Q = 0.01/R_0$. The process noise Q is decreased to $0.003/R_0$ and $0.001/R_0$ at epoch numbers 601 and 1401, respectively. The 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 [23]. The final network is tested on many new time series 128,000 points long (examples of totally new quadratic functions) resulting in RMS errors of less that 0.025. Figure 1.1 illustrates typical behavior of the trained network on a test time series. Just after the function change occurs, the network makes relatively large errors. It requires presenting 50 examples of the new function to the network to reduce the error to an acceptable level.

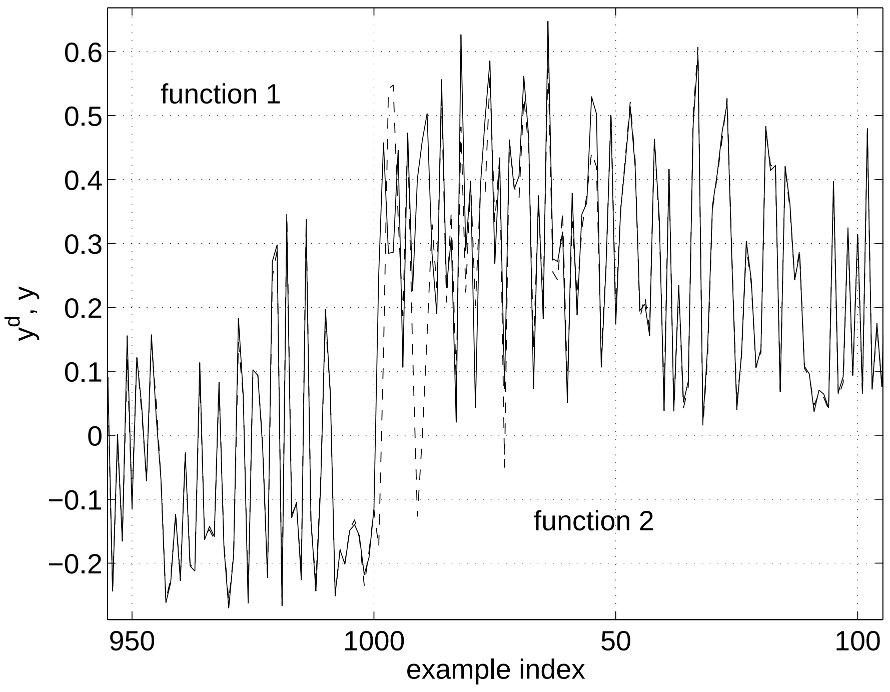

Fig. 1.1 Typical behavior of the trained network during testing. A fragment of the test time series with two different quadratic functions is shown. The target is solid, and the network output is dashed. The function change is clearly visible. The transient subsides within 50 example presentations.

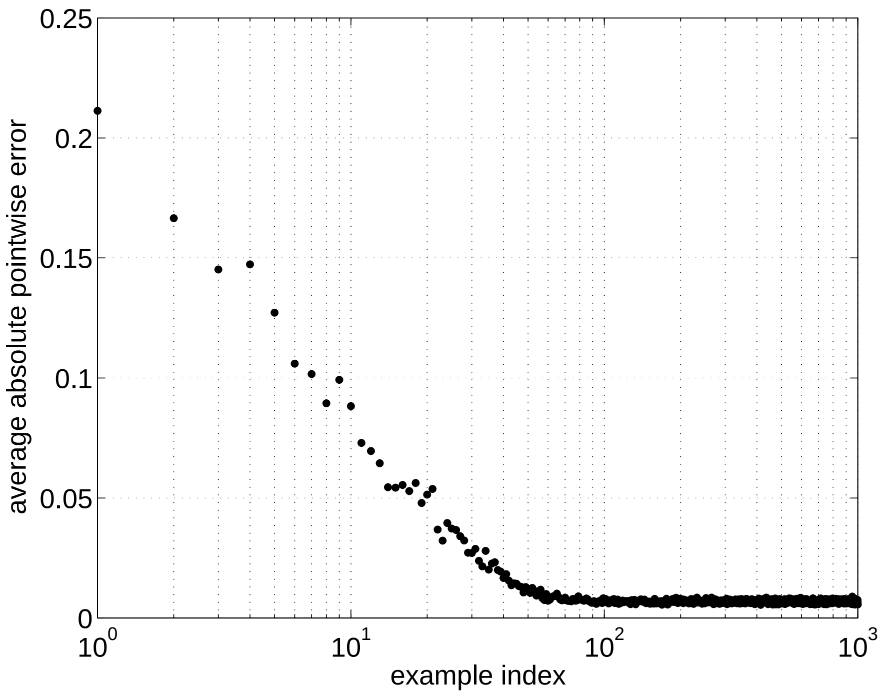

Fig. 1.2 The absolute pointwise error averaged over 128 functions for the 1000-example segment for a typical test time series. For each point in 1000-example segment the absolute instantaneous error was computed and averaged over all 128 functions.

We observe that, for this particular problem, node-decoupled EKF training seems to result in much worse performance than that of the global EKF-trained RMLP (the gradient descent seems utterly hopeless). Likewise, a significantly shorter truncation depth h of BPTT(h) or a substantially smaller RNN also appears to be insufficient to deliver acceptable performance.

In Figure 1.2 we show the absolute instantaneous error averaged over 128 functions for the 1000-example segment. The average error is about 0.1 after presentation of 10 examples. It decreases significantly to about 0.01 after presentation of 100 examples. It is clear that the network spends a fairly significant number of examples to figure out what function it deals with, but eventually results in a good steady state solution.

It is interesting to contemplate application of DAC to this problem. First, we note that it may be necessary to have a critic with as many as 40 outputs (one per each recurrent node of the RMLP). Second, we may need to use more powerful procedures for both the critic and the network training because the gradient descent does not suffice. And we also need to decide how to treat the coefficients a, b, c, d, e, f: whether 1) to use them as inputs to the critic, or 2) interpret them as disturbances and ignore them, or 3) use a separate critic for each type of quadratic functions.

The second option might result in an insufficiently accurate critic, whereas the third option could be impractical. The first option offers a tradeoff between critic accuracy and complexity of training.

1.7 CONCLUSION

We demonstrate that BPTT and DAC are closely related. This enables us to establish a common framework for comparison of the two methods featuring 1) analysis of critic representation, 2) a hybrid for smooth integration of BPTT and DAC, and 3) computational cost comparison. In our framework, both methods are considered in application to a heterogeneous recurrent network subsuming essential modules of the control system, i.e., plant, its model, reference model and feedback controller. This viewpoint is immediately applicable to modeling problems for which the use of RNN is advantageous. We also propose avenues for future comparative studies.

Acknowledgments

The author is very grateful to Dr. Lee Feldkamp for many helpful discussions and his past contributions to research on this topic.

REFERENCES

1. P. J. Werbos. “Backpropagation through time: What it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
2. R. J. Williams and D. Zipser, “Gradient-based learning algorithms for recurrent networks and their computational complexity,” in Chauvin and Rumelhart (Eds.), *Backpropagation: Theory, Architectures and Applications*, New York: L. Erlbaum and Associates, pp. 433–486, 1995.
3. P. J. Werbos, “Approximate dynamic programming for real-time control and neural modeling,” in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. A. White and D. A. Sofge, Eds., pp. 493–525, 1992.
4. D. V. Prokhorov, Adaptive Critic Designs and their Applications. Ph.D. dissertation, Department of Electrical Engineering, Texas Tech University, Lubbock, TX, October 1997.
5. D. V. Prokhorov and D. C. Wunsch, “Adaptive Critic Designs,” *IEEE Trans. Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.
6. D. V. Prokhorov, “Optimal Neurocontrollers for Discretized Distributed Parameter Systems,” in *Proceedings of the American Control Conference*, Denver, CO, 2003, pp. 549–554.

7. L. A. Feldkamp and D. V. Prokhorov, “Observations on the Practical Use of Adaptive Critics,” in *Proceedings of the 1997 IEEE SMC Conference,* Orlando, FL, 1997, pp. 3061–3066.

8. K. S. Narendra and A. M. Annaswamy. *Stable Adaptive Systems.* Prentice Hall, 1989.

9. D. V. Prokhorov and L. A. Feldkamp, “Primitive Adaptive Critics,” in *Proceedings of the 1997 IEEE International Conference on Neural Networks,* Houston, TX, 1997, pp. IV-2263–2267.

10. E. A. Wan, “Temporal Backpropagation for FIR neural networks,” in *Proceedings of the 1990 International Joint Conference on Neural Networks,* San Diego, 1990, pp. I-575–580.

11. L. A. Feldkamp and D. V. Prokhorov, “Phased Backpropagation: A Hybrid of TB and BPTT,” in *Proceedings of the International Joint Conference on Neural Networks,* Anchorage, AK, 1998, pp. 2262–2267.

12. L. A. Feldkamp, D. V. Prokhorov, C. F. Eagen, and F. Yuan, “Enhanced multi-stream Kalman filter training for recurrent networks,” in J. Suykens and J. Vandewalle (eds), *Nonlinear Modeling: Advanced Black-Box Techniques*, pp. 29–53. Kluwer Academic Publishers, 1998.

13. L. A. Feldkamp and G. V. Puskorius, “A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering and classification,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2259–2277, 1998.

14. L. A. Feldkamp, G. V. Puskorius, and D. V. Prokhorov, “Unified Formulation for Training Recurrent Networks with Derivative Adaptive Critics,” in *Proceedings of the 1997 International Conference on Neural Networks*, Houston, TX, 1997, pp. IV-2268–2272.

15. G. Lendaris and C. Paintz, “Training Strategies for Critic and Action Neural Nets in Dual Heuristic Programming Method," in *Proceedings of International Conference on Neural Networks,* Houston, TX, June, 1997, pp. 712–717.

16. G. Lendaris, T. T. Shannon, and A. Rustan, “A Comparison of Training Algorithms for DHP Adaptive Critic Neuro-control," in *Proceedings of International Conference on Neural Networks (IJCNN’99),* Washington,DC, July, 1999.

17. E. S. Plumer, “Optimal Control of Terminal Processes Using Neural Networks," *IEEE Trans. Neural Networks,* Vol. 7, No. 2, 1996, pp. 408–418.

18. R. Stengel. *Optimal Control and Estimation.* Dover, 1994.

19. S. W. Piché, “Steepest Descent Algorithms for Neural Network Controllers and Filters," *IEEE Trans. Neural Networks*, Vol. 5, No. 2, 1994, pp. 198–212.

20. *Nonlinear Model Predictive Control*, F. Allgöwer and A. Zheng (Eds.), Progress in systems and Control Theory Series, Vol. 26, Birkhauser Verlag, Basel 2000.

21. D. V. Prokhorov and L. A. Feldkamp, “Analyzing for Lyapunov Stability with Adaptive Critics,” in *Proceedings of the 1998 IEEE Conference on Systems, Man, and Cybernetics,* San Diego, CA, 1998, pp. 1658–1661.

22. L. A. Feldkamp, T. M. Feldkamp, and D. V. Prokhorov. “Recurrent Neural Network Training by nprKF Joint Estimation.” *Proceedings of International Joint Conference on Neural Networks ’02*, Hawaii, 2002.

23. D. Prokhorov, L. Feldkamp, and I. Tyukin, “Adaptive Behavior with Fixed Weights in RNN: Overview." *Proceedings of International Joint Conference on Neural Networks ’02*, Hawaii, 2002.

24. 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.

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

26. L. A. Feldkamp and G. V. Puskorius, “Fixed weight controller for multiple systems,” *Proceedings of the 1997 International Joint Conference on Neural Networks*, Houston TX, 1997, vol. II, pp. 773–778.

27. L. A. Feldkamp, G. V. Puskorius, and P. C. Moore, “Adaptive behavior from fixed weight networks,” *Information Sciences*, vol. 98, pp. 217–235, 1997.

28. L. Feldkamp, D. Prokhorov, and T. Feldkamp, “Simple and Conditioned Adaptive Behavior from Kalman Filter Trained Recurrent Neural Network," *Neural Networks*, vol. 16, pp. 683–689, 2003.

29. D. V. Prokhorov, G. V. Puskorius, and L. A. Feldkamp, “Dynamical Neural Networks for Control," see in *A Field Guide to Dynamical Recurrent Networks*, J. Kolen and S. Kremer (Eds.), IEEE Press, 2001, pp. 257–289.

30. D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch. “Adaptive Critic Designs: A Case Study For Neurocontrol," *Neural Networks*, vol. 8, no. 9, pp. 1367–1372, 1995.

31. S. N. Balakrishnan and V. Biega, “Adaptive Critic Based Neural Networks for Aircraft Optimal Control," *Journal of Guidance, Control and Dynamics,* Vol. 19, No. 4, pp. 893–898, 1996.

32. G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, “Comparison of Heuristic Dynamic Programming and Dual Heuristic Programming Adaptive Critics for Neurocontrol of a Turbogenerator,” *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 764–773, 2002.

33. P. H. Eaton, D. V. Prokhorov, and D. C. Wunsch, “Neurocontroller Alternatives for “Fuzzy" Ball-and-Beam Systems with Nonuniform Nonlinear Friction," *IEEE Trans. Neural Networks*, vol. 11, no. 2, pp. 423–435, 2000.