A Constrained Optimization Approach to Preserving Prior Knowledge During Incremental Training

Silvia Ferrari, *Member, IEEE*, and Mark Jensenius, *Member, IEEE*

***Abstract*—In this paper, a supervised neural network training technique based on constrained optimization is developed for preserving prior knowledge of an input–output mapping during repeated incremental training sessions. The prior knowledge, referred to as long-term memory (LTM), is expressed in the form of equality constraints obtained by means of an algebraic training technique. Incremental training, which may be used to learn new short-term memories (STMs) online, is then formulated as an error minimization problem subject to equality constraints. The solution of this problem is simplified by implementing an adjoined error gradient that circumvents direct substitution and exploits classical backpropagation. A target application is neural network function approximation in adaptive critic designs. For illustrative purposes, constrained training is implemented to update an adaptive critic flight controller, while preserving prior knowledge of an established performance baseline that consists of classical gain-scheduled controllers. It is shown both analytically and numerically that the LTM is accurately preserved while the controller is repeatedly trained over time to assimilate new STMs.**

***Index Terms*—Adaptive critics, control, exploration, function approximation, incremental training, interference, knowledge acquisition and retention, memory, online learning, sigmoidal neural networks.**

I. Introduction

THE ability to preserve prior knowledge while processing and learning new information is crucial for neural network applications such as adaptive critics, control, and system identification. We define as long-term memory (LTM) the prior knowledge that must be preserved by an artificial neural network at all times. New information that is not consolidated into LTM is referred to as short-term memory (STM). It is well known that when neural networks are trained incrementally with new data, functional relationships acquired from previous training sets deteriorate through a process known as *interference*. Various solutions have been proposed to address the problem. One approach presents some of the LTM and STM data together to suppress interference in supervised incremental training algorithms [1]–[3]. While very effective for some applications, it is not suitable for those that require highly accurate preservation of LTM, or that have stringent computational requirements due, for example, to large input–output spaces, or to learning being performed online. Another approach consists of using extra hidden units to augment the approximation power of the neural network, and of partitioning the weights into two subsets, referred to as LTM and STM weights. The LTM weights are used to preserve LTM by holding their values constant, while the STM weights are updated with the new STM data [4], [5]. The latter approach is more consistent with the memory-formation mechanisms observed in the nervous system [6]. However, due to nonlinearity and global support properties, modifying the STM weights in a sigmoidal neural network can unfortunately also significantly change its input–output mapping everywhere in its domain. Therefore, holding the LTM weights constant cannot always guarantee accurate LTM preservation, and is particularly ineffective when training sigmoidal neural networks.

Similarly to Mandziuk and Kotani [4], [5], the constrained training approach presented in this paper also partitions the weights into LTM and STM subsets. However, in our approach, both subsets are updated: the first to meet the LTM equality constraints, and the second to minimize the STM network error through a constrained optimization technique. The keystones of this novel approach are the formulation of the equality constraints by means of algebraic training, and the simplification of the resulting constrained optimization problem by means of the adjoined error gradient.

One target application for this constrained-training approach is neural network function approximation in adaptive critic designs [7]. Using prior knowledge of classical controllers to obtain the starting design is a key step in the development of highly effective adaptive neural controllers [8]–[17]. However, due to interference, when such controllers undergo prolonged adaptation, they may experience a significant loss in baseline performance and possibly even catastrophic forgetting [18], as shown in Section V-C. Consequently, these controllers may need to relearn how to control the plant online, even under conditions for which the classical controllers are optimal and known *a priori*. In this paper, constrained training is applied to an adaptive critic flight controller presented in [17], in order to preserve LTM of classical gain-scheduled controllers. These classical controllers are locally optimal and provide a satisfactory baseline performance when the aircraft operates in its steady-level flight envelope. When the aircraft operates outside this envelope, the neural network controller adapts to compensate for nonlinear or unmodeled dynamics. Through constrained training, the neural network controller simultaneously retains accurate memory of the classical gain-scheduled designs and, consequently, displays

Manuscript received August 29, 2006; revised May 15, 2007 and August 15, 2007; accepted September 23, 2007. This work was supported by the National Science Foundation under Grant ECS 0300236 and CAREER award ECS 0448906.

S. Ferrari is with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708-0005 USA (e-mail: sferrari@duke.edu).

M. Jensenius is with the Applied Physics Laboratory, The Johns Hopkins University, Laurel, MD 20723 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2007.915108

optimal performance as soon as the aircraft returns to its steady-level flight envelope.

The constrained training approach is presented in Section III. The derivation of the equality constraints and adjoined error gradient are illustrated for a gradient-based LTM training set in Section IV. Section V-D presents the simulation results for the constrained adaptive critic flight controller, which implements the control design presented in [17] and reviewed in Section V-A.

II. Background on Supervised Neural Network Training

Sigmoidal neural networks are used in a variety of applications because they are universal function approximators that can learn unknown functional relationships by example, in batch or in incremental mode. Supervised training involves the minimization of the error between the available data, referred to as *training set*, and the actual network performance with respect to its adjustable parameters or *weights*. Consider a training set $\mathcal{T} = \{\mathbf{y}^l, \mathbf{u}^l\}_{l=1,\dots,p}$ of error-free input–output values of an unknown function $\mathbf{u} = h(\mathbf{y})$, where $\mathbf{u} \in \mathcal{R}^m$ and $\mathbf{y} \in \mathcal{R}^r$. In supervised *batch training*, the error function is defined with respect to the entire training set $\mathcal{T}$. In *incremental training*, the network error function to be minimized is defined with respect to one training sample at a time

$$\begin{aligned} &\text{minimize} \quad e_l(\mathbf{U}) \equiv \frac{1}{2}(\mathbf{z}^l - \mathbf{u}^l)^T(\mathbf{z}^l - \mathbf{u}^l) \\ &\text{subject to} \quad \mathbf{U} \in \mathcal{R}^N \end{aligned} \tag{1}$$

where $\mathbf{z}^l$ denotes the network output for the input $\mathbf{y}^l$, and the N adjustable weights $\mathbf{U}$ are updated through p training sessions. During each training session, an unconstrained optimization algorithm adjusts the weights iteratively based on the error gradient

$$\mathbf{U}^{(\ell+1)} = \mathbf{U}^{(\ell)} - \eta^{(\ell)} \left.\frac{\partial e_l}{\partial \mathbf{U}}\right|_{(\ell)} \tag{2}$$

where ℓ is the *epoch* index, and η is the learning rate. Levenberg–Marquardt and resilient backpropagation (RPROP) algorithms typically exhibit the fastest convergence thanks to the techniques by which they vary the learning rate [19], [20].

The convergence of the above incremental gradient algorithm has been studied extensively in the literature [21], [22]. In fact, incremental training is widely used in practice, either in lieu of or in combination with batch training, because it can handle large training sets that are otherwise prohibitive even for moderate-size networks [23]–[27]. Moreover, incremental training allows to train neural networks online, assimilating training samples that only become available one at a time incrementally over time. It is well known, however, that prior functional knowledge tends to be destroyed by conventional incremental training algorithms due to interference. The following section introduces a constrained-optimization training technique that allows to preserve prior functional knowledge while learning from new data incrementally.

III. Constrained Neural Network Training

Neuroscientists have long speculated that, in biological organisms, learning alters connections between neurons, thereby forming and maintaining new memories. Recent evidence supports the theory that STM formation is accompanied by the activation of an existing set of potential synaptic connections that are transformed into functional connections [6]. Using this as an analogy, in constrained training, potential synaptic connections are represented by STM weights that are initially set to zero, and subsequently adjusted to learn new STM information. Because sigmoidal neural networks are characterized by global support, once the STM weights are updated, LTM functional knowledge cannot be preserved simply by holding the LTM weights constant. Therefore, LTM is preserved by formulating supervised training as a constrained optimization problem. The theoretical foundations of this constrained training approach are presented in Section III-A, and its implementation is discussed in Section III-B.

A. Foundations of Constrained Training

Typically, supervised training is formulated as an unconstrained optimization problem involving a scalar error function of many variables, consisting of the network weights $\mathbf{U}$ (Section II). Assume the LTM functional knowledge can be embedded into a relationship describing the network weights such as

$$g(\mathbf{L}, \mathbf{S}) = \mathbf{0} \tag{3}$$

where $\mathbf{U}$ has been reorganized into a matrix of LTM weights $\mathbf{L} \in \mathcal{R}^{K\times M}$ and a matrix of STM weights $\mathbf{S} \in \mathcal{R}^{P\times Q}$ with $KM + PQ = N$. If (3) satisfies the implicit function theorem (Appendix I), then it uniquely implies the function

$$\mathbf{L} = \mathcal{C}(\mathbf{S}) \tag{4}$$

which can be used instead of (3) to simplify the training problem. Training preserves the LTM expressed by (3) provided it is carried out according to the following constrained optimization problem:

$$\begin{aligned} &\text{minimize} \quad e_l(\mathbf{L}, \mathbf{S}) \equiv \frac{1}{2}(\mathbf{z}^l - \mathbf{u}^l)^T(\mathbf{z}^l - \mathbf{u}^l) \\ &\text{subject to} \quad g(\mathbf{L}, \mathbf{S}) = \mathbf{0}, \qquad \mathbf{L} \in \mathcal{R}^{K\times M}, \qquad \mathbf{S} \in \mathcal{R}^{P\times Q}. \end{aligned} \tag{5}$$

In the remainder of this paper, it is assumed that the STM is acquired through incremental training. Therefore, the error function e_l is abbreviated to e, for simplicity. Then, the approach described below can be applied to constrain a batch training algorithm, simply by modifying the definition of the error function e.

The solution of a constrained optimization problem can be provided by the method of Lagrange multipliers or by direct elimination [28], [29]. If the equality constraints can be written explicitly (4), the method of direct elimination can be applied by writing the error function as

$$E(\mathbf{S}) = e\left(\mathcal{C}(\mathbf{S}), \mathbf{S}\right) \tag{6}$$

such that the value of $\mathbf{S}$ can be determined independently of $\mathbf{L}$. In this case, the solution of (5) is an extremum of (6) that obeys

$$\frac{\partial E}{\partial \mathbf{S}} = \mathbf{0}. \tag{7}$$

Once the optimal value of $\mathbf{S}$ is determined, the optimal value of the weights $\mathbf{L}$ can be obtained from $\mathbf{S}$ using (4). If the equality constraint cannot be written as (4), the method of Lagrange multipliers can be used to solve (5) [28], [29].

Hereon, it is assumed that the equality constraint can be written in explicit form (4). Furthermore, because (4) can be very involved, its substitution in the error function is circumvented by seeking the extremum defined by the *adjoined error gradient*, obtained by the chain rule

$$\frac{\partial E}{\partial \mathbf{S}} \equiv \left\{ \frac{\partial E}{\partial s_{ij}} \right\} = \left\{ \frac{\partial e}{\partial s_{ij}} + \frac{\partial e}{\partial l_{rt}} \frac{\partial l_{rt}}{\partial s_{ij}} \right\} = \partial_{\mathbf{S}} e + \mathcal{G}[\mathbf{L}, \mathbf{S}, \partial_{\mathbf{L}} e]. \tag{8}$$

In tensor notation, matrices are written in terms of their elements, such that $\mathbf{L} = \{l_{rt}\}$ and $\mathbf{S} = \{s_{ij}\}$, and repetition of an index within any term in an equation indicates summation over that index. Tensor notation is employed in this paper for its compactness. Also, the shorthand notation $\partial_{\mathbf{A}} J$ is used to denote the gradient of a scalar function J with respect to an $(n \times m)$ matrix $\mathbf{A}$, which is defined as the $(n \times m)$ matrix of partial derivatives $\{\partial J / \partial a_{ij}\}$.

Using the adjoined gradient, the effect of the LTM weights $\mathbf{L}$ on the constrained network error E is accounted for, and E can be minimized solely with respect to $\mathbf{S}$. Using the properties described in Appendix II, the adjoined error gradient can be obtained in terms of the derivatives $\partial_{\mathbf{S}} e$ and $\partial_{\mathbf{L}} e$, which are easily computed by classical backpropagation.

B. Constrained Training Implementation

The constrained training approach is applicable to incremental training of neural networks for smooth function approximation under the following assumptions: 1) *a priori* knowledge of the function is available locally in its domain (e.g., in the form of a batch training set or a physical model); 2) it can be expressed as an equality constraint on the neural network weights; 3) it is desirable to preserve this prior knowledge during future training sessions; 4) new functional information must be assimilated incrementally through domain exploration; and 5) the new information is consistent with the prior knowledge (i.e., the function to be approximated is one-to-one and the information is noise free). Then, the constrained training implementation can be summarized by the following steps.

1) Determine the equality constraint equation (3) for the chosen neural network architecture.
2) Determine the neural network size, and the LTM–STM connections $\mathbf{S}$ and $\mathbf{L}$.
3) Rewrite the equality constraints in explicit form (4).
4) Compute the adjoined error gradient (8) analytically, using its properties (Appendix II).
5) Apply the constrained training algorithm (Algorithm 1).

The following algorithm trains a neural network incrementally within a user-set tolerance e_{tol}. Simultaneously, it retains the functional knowledge expressed by the equality constraint (3). The update of the STM weights is conducted by a gradient-based training algorithm (RPROP).

Algorithm 1: Constrained Neural Network Training {

given $e(\cdot)$, $\mathcal{C}(\cdot)$, and $\partial E / \partial \mathbf{S}(\cdot)$

compute $\mathbf{L}^{(0)}$ by unconstrained training, such that $\mathbf{S}^{(0)} = \mathbf{0}$, and $g(\mathbf{L}^{(0)}, \mathbf{0}) = 0$

for $l = 1 : p$,

$\ell = 0$

while $e(\mathbf{L}, \mathbf{S}) \equiv (1/2)(\mathbf{z}^l - \mathbf{u}^l)^T (\mathbf{z}^l - \mathbf{u}^l) > e_{\text{tol}}$,

update the STM weights:
$\mathbf{S}^{(\ell+1)} = \mathbf{S}^{(\ell)} - \eta^{(\ell)} \partial E / \partial \mathbf{S}|_{(\ell)}$

update the LTM weights:
$\mathbf{L}^{(\ell+1)} = \mathcal{C}(\mathbf{S}^{(\ell+1)})$

$\ell = \ell + 1$

end

$\mathbf{L}^{(0)} = \mathbf{L}^{(\ell+1)}$, and $\mathbf{S}^{(0)} = \mathbf{S}^{(\ell+1)}$

end

}

The equality constraint (3) for a generic LTM training set can be determined through the algebraic training technique presented in [30]. In other applications, (3) may be determined from application-specific knowledge of the function to be approximated or its properties. For example, control law properties such as stability can be formulated through linear matrix equalities and inequalities on the neural weights [31], [32]. Also, in system identification by gray-box models, the constraint (3) can be determined from well-known physical models with limited domain applicability [33], [34].

One approach for selecting LTM and STM connections is to size the network through a trial-and-error procedure. A common practice is to enhance the neural network approximation abilities by adding nodes to its hidden layer. Thus, after having determined the number of nodes required to properly approximate the LTM, the user can introduce additional nodes and denote the corresponding connections by $\mathbf{S}$. Another approach is to employ algebraic training [30] to determine both the number of nodes and the LTM-STM connections, as illustrated in the following section.

IV. Constrained Training Equations for Gradient-Based LTM

The main difference between the implementation of constrained training versus classical backpropagation is the computation of the error gradient. In this section, the constraints (3) and the corresponding adjoined gradient are obtained for the gradient-based LTM training set presented in [30]. As shown in [30], this gradient-based training set affords excellent generalization properties and may be used to represent gain-scheduled

controllers or system models at multiple equilibria [16], as shown by the example provided in Section V-A.

Consider the class of one-layer, s-node sigmoidal neural networks

$$\mathbf{z}(\mathbf{y}) = \mathbf{V}\Phi(\mathbf{W}\mathbf{y}) + \mathbf{b} \tag{9}$$

where $\mathbf{z} \in \mathcal{R}^m$, $\mathbf{y} \in \mathcal{R}^r$, $\mathbf{b}$ is the output bias, and $\Phi(\cdot)$ is an operator with repeated sigmoidal nonlinearities $\sigma(n) \equiv (e^n - 1)/(e^n + 1)$ (the input bias is set equal to zero for simplicity). Let the input be partitioned into a *regular* input vector $\mathbf{x} \in \mathcal{R}^n$, for which derivative information may be available, and an *auxiliary* input vector $\mathbf{a} \in \mathcal{R}^\nu$, for which function derivatives are unknown, i.e., $\mathbf{y} = [\mathbf{x}^T \quad \mathbf{a}^T]^T$. Depending on the application, the Jacobian

$$\mathbf{C}_{ij} \equiv \frac{\partial \mathbf{u}_i}{\partial \mathbf{x}_j} \tag{10}$$

may be known for selected outputs and regular inputs indexed by i and j, respectively. Then, as shown in [30], a set $\mathcal{T}_{\text{LTM}} = \{\mathbf{y}^k, \mathbf{u}^k, \mathbf{C}_{ij}^k\}_{k=1,\dots,q}$, with $\mathbf{y}^k = [\mathbf{0}_{1\times n}\ (\mathbf{a}^k)^T]^T$, can be used to approximate a nonlinear function by a sigmoidal neural network based primarily on derivative information. In this example, $\mathbf{C}_{ij}$ is known for $(i,j) = (1,1)$ and $(2,2)$, where

$$\mathbf{x} = \begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix} = \{\mathbf{x}_j\} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{bmatrix} = \{\mathbf{u}_i\}, \qquad i,j = 1,2. \tag{11}$$

The equality constraints are then obtained using algebraic training [30] and the positional notation described in Appendix III, as shown by the following result.

Theorem 1 (Equality Constraints): A gradient-based training set $\mathcal{T}_{\text{LTM}} = \{\mathbf{y}^k, \mathbf{u}^k, \mathbf{C}_{11}^k, \mathbf{C}_{22}^k\}_{k=1,\dots,q}$, with $\mathbf{y}^k = [\mathbf{0}_{1\times n}\ (\mathbf{a}^k)^T]^T$, can be embedded in a neural network (9) satisfying the equality constraints

$$\begin{cases} \mathbf{V}(\Theta_i, H_1)\mathbf{\Sigma}(H_1) + \mathbf{V}(\Theta_i, H_2)\mathbf{\Sigma}(H_2) \\ \quad +\mathbf{B}(\Theta_i) - \mathbf{Z}_i = \mathbf{0}, \qquad i = 1,2 \\ \mathbf{V}(\Theta_i, H_1)\mathbf{\Lambda}^k(H_1)\mathbf{W}_{\mathbf{x}}(H_1, I_j) \\ \quad +\mathbf{V}(\Theta_i, H_2)\mathbf{\Lambda}^k(H_2)\mathbf{W}_{\mathbf{x}}(H_2, I_j) - \mathbf{C}_{ij}^k = \mathbf{0}, \\ (i,j) = (1,1),(2,2), \qquad k = 1,\dots,q \end{cases} \tag{12}$$

in the implicit form (3), where

$$\begin{bmatrix}\mathbf{Z}_1\\ \mathbf{Z}_2\end{bmatrix} \equiv \begin{bmatrix}\mathbf{u}_1^1 & \cdots \mathbf{u}_1^q\\ \mathbf{u}_2^1 & \cdots \mathbf{u}_2^q\end{bmatrix} \tag{13}$$

$$\mathbf{A} \equiv [\mathbf{a}^1 \cdots \mathbf{a}^q] \tag{14}$$

$$\mathbf{\Sigma} \equiv \Phi(\mathbf{W}_{\mathbf{a}}\,\mathbf{A}) \tag{15}$$

$$\mathbf{\Lambda}^k \equiv \operatorname{diag}\left[\Phi'(\mathbf{W}_{\mathbf{a}}\,\mathbf{a}^k)\right]. \tag{16}$$

$\mathbf{B}$ is an $(m \times q)$ matrix with q columns all equal to $\mathbf{b}$, and $\mathbf{W} = [\mathbf{W}_{\mathbf{x}}\ \mathbf{W}_{\mathbf{a}}]$.

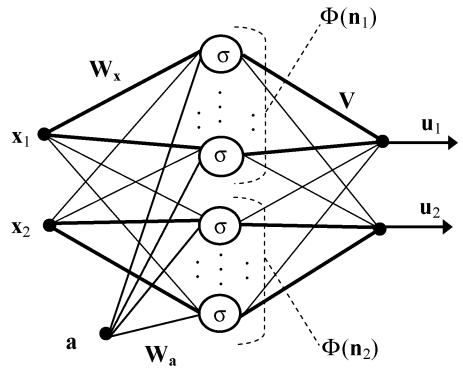

Fig. 1. Neural network architecture with LTM and STM connections represented by thick and thin lines, respectively.

The proof is provided in Appendix IV. Typically, the size of the network and the LTM–STM connections can be designed by inspection of the equality constraints. In this case, let $s = 2q$, and partition the hidden nodes into two q-node sets, with outputs $\Phi(\mathbf{n}_1)$ and $\Phi(\mathbf{n}_2)$. Choose the LTM–STM connections as illustrated in Fig. 1. Then, as shown by the following proposition, the constraints (12) can be written in the explicit form (4).

Proposition 1 (Explicit Constraints): The constraint equation (12) implies the function shown in (17) at the bottom of the page, where

$$\mathbf{K}_{ij} \equiv \begin{bmatrix}\mathbf{C}_{ij}^1\\ \vdots\\ \mathbf{C}_{ij}^q\end{bmatrix} \quad \mathbf{\Lambda} \equiv \begin{bmatrix}\mathbf{\Lambda}^1\\ \vdots\\ \mathbf{\Lambda}^q\end{bmatrix} \quad \text{and} \quad \mathbf{V}_d(\cdot) \equiv \operatorname{diag}\left[\mathbf{V}(\cdot)\right]. \tag{18}$$

The proof is provided in Appendix V. As in (4), the constraints (17) provide an explicit relationship between the LTM weights $\mathbf{L} \equiv [\mathbf{L}_1^T\ \mathbf{L}_2^T]^T$ and the STM weights $\mathbf{S} \equiv [\mathbf{S}_1^T\ \mathbf{S}_2^T]^T$, where $\mathbf{S}_\imath \equiv [\mathbf{V}(\Theta_\imath, \bar{H}_\imath)^T\ \mathbf{W}_{\mathbf{x}}(\bar{H}_\imath, I_\imath)]^T$, for $\imath = 1,2$. The direct substitution of (17) into the network error function (6) is circumvented by means of the adjoined error gradient. The adjoined error gradient, defined in (8), is derived in terms of the error gradients $\partial_{\mathbf{S}} e$ and $\partial_{\mathbf{L}} e$, which are easily obtained through classical backpropagation, as shown by the following result.

Theorem 2 (Adjoined Gradient): Let (5) define the error function for the network (9), subject to the constraint equation (17) on the network weights. Then, the adjoined gradient is given by

$$\frac{\partial E}{\partial \mathbf{S}} \equiv \left[\frac{\partial E}{\partial \mathbf{V}}(\Theta_1, \bar{H}_1)^T \quad \frac{\partial E}{\partial \mathbf{W}_{\mathbf{x}}}(\bar{H}_1, I_1) \quad \frac{\partial E}{\partial \mathbf{V}}(\Theta_2, \bar{H}_2)^T \quad \frac{\partial E}{\partial \mathbf{W}_{\mathbf{x}}}(\bar{H}_2, I_2)\right]^T \tag{19}$$

$$\mathbf{L}_\imath \equiv \begin{bmatrix}\mathbf{V}(\Theta_\imath, H_\imath)\\ \mathbf{W}_{\mathbf{x}}(H_\imath, I_\imath)^T\end{bmatrix} = \begin{bmatrix} -\left[\mathbf{B}(\Theta_\imath) - \mathbf{Z}_\imath + \mathbf{V}(\Theta_\imath, \bar{H}_\imath)\mathbf{\Sigma}(\bar{H}_\imath)\right] \cdot \mathbf{\Sigma}(H_\imath)^{-1} \\ \left[\mathbf{K}_{\imath\imath} - \mathbf{V}_d(\Theta_\imath, \bar{H}_\imath)\mathbf{\Lambda}(\bar{H}_\imath)\mathbf{W}_{\mathbf{x}}(\bar{H}_\imath, I_\imath)\right]^T \cdot \left[\mathbf{V}_d(\Theta_\imath, H_\imath)\mathbf{\Lambda}(H_\imath)\right]^{-T} \end{bmatrix} \tag{17}$$

$$\imath = 1,2$$

where

$$\frac{\partial E}{\partial \mathbf{V}}(\Theta_\imath, \bar{H}_\imath) = \partial_{\mathbf{V}} e(\Theta_\imath, \bar{H}_\imath) + \mathcal{G}_m\left[\mathbf{V}(\Theta_\imath, \bar{H}_\imath)\right] + \{\partial_{\mathbf{V}} e(\Theta_\imath, H_\imath) + \mathcal{G}_m\left[\mathbf{V}(\Theta_\imath, H_\imath)\right]\} \cdot \left[\mathbf{\Sigma}(\bar{H}_\imath)\mathbf{\Sigma}(H_\imath)^{-1}\right]^T$$

$$\frac{\partial E}{\partial \mathbf{W_x}}(\bar{H}_\imath, I_\imath) = \partial_{\mathbf{W_x}} e(\bar{H}_\imath, I_\imath) - \mathbf{\Lambda}^T(\bar{H}_\imath)\mathbf{V}_d(\Theta_\imath, \bar{H}_\imath) \cdot \left[\mathbf{\Lambda}^T(H_\imath)\mathbf{V}_d(\Theta_\imath, H_\imath)\right]^{-1} \partial_{\mathbf{W_x}} e(H_\imath, I_\imath)$$

$\imath = 1, 2$, and the function $\mathcal{G}_m[\mathbf{V}(\cdot)]$ is defined as in (60).

The proof is provided in Appendix VI. The remaining sections describe an important application of the constrained training approach, namely, neural network function approximation in adaptive critic designs. The theoretical results derived above are validated numerically in Section V-D. In these simulations, the adjoined gradient presented in Theorem 2 is implemented, and the results show that $\mathcal{T}_{\text{LTM}}$ is preserved accurately during every incremental training session.

V. Application: Constrained-Training of an Adaptive Critic Flight Controller

A target application of the constrained-training approach developed in this paper is neural network function approximation in adaptive critic designs. Adaptive critics approximate solutions to complex optimal control problems by optimizing neural approximations of the control law and value function iteratively over time [7]. Lendaris and Neidhoefer write that "if *a priori* knowledge is available about the problem domain that may be translated into a starting design of the controller and/or the critic, then it behooves us to use this knowledge as a starting point for the ADP procedures" [12]. One reason is that the design may exploit prior knowledge to perform adequately while the ADP system is learning online. Another reason is that if the starting design is close to the optimal solution, then the ADP system works on refining the existing design instead of learning from scratch. Even when prior knowledge is successfully embedded in the starting adaptive critic controller, it may be deteriorated or completely forgotten after frequent parameter updates in the exploration phase.

In this section, the constrained-training approach is implemented to preserve an adaptive critic flight controller's knowledge of a set of classical gain-scheduled designs. The simulated controller, taken from [17], is reviewed in Sections V-A and V-B. In Section V-C, it is shown that, after exploring regimes that are significantly nonlinear, memory of the gain-scheduled designs is considerably degraded due to interference. Finally, in Section V-D, the constrained training approach is implemented, thereby suppressing interference and preserving accurate memory of these designs at all times.

A. Review of Adaptive Critic Flight Control Design

It is assumed that the aircraft dynamics can be captured by a nonlinear differential equation

$$\dot{\mathbf{x}}(t) = f\left[\mathbf{x}(t), \mathbf{p}_m(t), \mathbf{u}(t)\right], \qquad \mathbf{x}(0) = \mathbf{x}_0 \tag{20}$$

where $\mathbf{x} \in \mathcal{X} \subset \mathcal{R}^n$ is the state, $\mathbf{u} \in \mathcal{U} \subset \mathcal{R}^m$ is the control, $\mathbf{x}_0$ is given, and $\{\mathcal{X}, \mathcal{U}\}$ is the *full* operating envelope of the aircraft. The ideal closed-loop response of a piloted aircraft is specified by well-known flying qualities criteria that reflect the pilot's impression of the aircraft [35]. These criteria can be used to formulate the control performance objectives as a cost function of the type

$$J = \int_0^{t_f} \mathcal{L}\left[\mathbf{x}(\tau), \mathbf{u}(\tau)\right] d\tau = \int_0^{t_f} \left[\tilde{\mathbf{x}}^T(\tau)\mathbf{Q}\tilde{\mathbf{x}}(\tau) + 2\tilde{\mathbf{x}}^T(\tau)\mathbf{M}\tilde{\mathbf{u}}(\tau) + \tilde{\mathbf{u}}^T(\tau)\mathbf{R}\tilde{\mathbf{u}}(\tau)\right] d\tau \tag{21}$$

with weighting matrices $\mathbf{Q}$, $\mathbf{M}$, and $\mathbf{R}$ designed using implicit model following (IMF) [28]. State and control deviations from the set point, denoted by $\tilde{(\cdot)}$, are used for command input or trajectory tracking. The objective is to determine a state-feedback neural network controller in the form (9) that optimizes (21) subject to the nonlinear dynamic equation (20) over $\mathcal{X}$.

1) Classical Gain-Scheduled Flight Controllers (or LTM): The classical gain-scheduled flight controllers reviewed in this section constitute the LTM to be preserved at all times by the neural network controller. These gain-scheduled controllers have been shown to be very effective for controlling aircraft dynamics over the steady-level flight envelope [35], [36]. Near steady-level flight conditions, the aircraft dynamics can be closely approximated by a class of affine systems

$$\Delta\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x})\Delta\mathbf{x} + \mathbf{G}(\mathbf{x})\Delta\mathbf{u} \tag{22}$$

that evolve on a subset of the state–space $\mathcal{X}_{\text{LPV}} \subset \mathcal{X}$, referred to as the linear parameter varying (LPV) regime. Δ denotes perturbations from the equilibrium. The functions $\mathbf{F}$ and $\mathbf{G}$ can be approximated by two sets of matrices $\{\mathbf{F}^k \approx \mathbf{F}(\mathbf{a}^k)\}$ and $\{\mathbf{G}^k \approx \mathbf{G}(\mathbf{a}^k)\}$ for q equilibria or *scheduling vectors* $\mathcal{P} = \{\mathbf{a}^1, \ldots, \mathbf{a}^q\} \in \mathcal{X}_{\text{LPV}}$. Also, for small perturbations, the aircraft dynamics can be decoupled into longitudinal (L) and lateral–directional (LD) modes of motion [35].

A well-known result in control theory states that when the cost function is quadratic and the dynamic equation is linear, the optimal control law takes the form

$$\Delta\mathbf{u}^*(t) = -\mathbf{R}^{-1}\left[\mathbf{G}^T\mathbf{P} + \mathbf{M}^T\right]\Delta\mathbf{x}^*(t) \equiv -\mathbf{C}\Delta\mathbf{x}^*(t) \tag{23}$$

where the system matrix $\mathbf{G}$, the weighting matrices $\mathbf{R}$ and $\mathbf{M}$, and the Riccati matrix $\mathbf{P}$, defined according to [28], are all known. Using (23), it is possible to obtain a set of longitudinal and lateral control gains that are locally optimal in $\mathcal{X}_{\text{LPV}}$ and are scheduled by $\mathbf{a}^k$, i.e., $\{\mathbf{C}_L^k, \mathbf{C}_{LD}^k\}_{k=1,\ldots,q}$ [16]. This set is used to form the LTM training set $\mathcal{T}_{\text{LTM}}$ in Section V-D.

2) Online Learning and Short-Term Control Knowledge: Adaptive critics aim at overcoming the well-known curse of dimensionality [38] by embedding the optimization of the cost

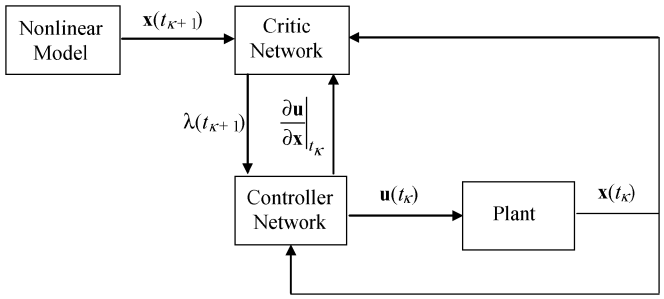

Fig. 2. Block diagram of DHP adaptive critic architecture implemented at time t_κ.

function (21) into the optimization of a value function V, and by discretizing the time interval $t \in [0,\ t_f)$ such that

$$V\left[\mathbf{x}(t_\kappa), c(\cdot)\right] \equiv \sum_{t_\kappa}^{t_f-\Delta t} \mathcal{L}_D\left[\mathbf{x}(t_\kappa), \mathbf{u}(t_\kappa)\right]. \tag{24}$$

Howard [39] showed that if the control law and value function approximations are updated incrementally over time by a policy-improvement routine and a value-determination operation, they eventually converge to their optimal counterparts $\mathbf{u}^* = c^*(\mathbf{x}^*)$ and $V^*[\mathbf{x}^*, c^*]$, respectively. Furthermore, at each iteration t_κ, these two approximations are improved and are closer to optimal than their predecessors.

In dual heuristic programming (DHP), the critic approximates the controller performance sensitivity to the state $\boldsymbol{\lambda} \equiv \partial V/\partial \mathbf{x}$, in place of V, to accelerate convergence. The ideal control performance over $\mathcal{X}$ is stated through the optimality condition presented in [40]

$$\frac{\partial \mathcal{L}_D}{\partial \mathbf{u}}\bigg|_{t_\kappa} + \frac{\partial f_D}{\partial \mathbf{u}}\bigg|_{t_\kappa}^{T} \boldsymbol{\lambda}(t_{\kappa+1}) = \mathbf{0} \tag{25}$$

where $f_D[\cdot]$ is the discretized form of (20). This equation constitutes a local condition for optimality that must be satisfied along the system trajectory. A recurrence relation for the critic is obtained by differentiating (24) with respect to the state, as shown in [15]. The DHP architecture is illustrated in Fig. 2. The critic neural network computes $\boldsymbol{\lambda}(t_{\kappa+1})$, required by (25), based on the predicted value of the state $\mathbf{x}(t_{\kappa+1})$ obtained from (20). Additional design and implementation details are presented in [17]. Because DHP produces sequences of functions that are progressively closer to optimal, the adaptation improves the local control performance with respect to the starting design, provided here by the gain-scheduled controllers reviewed in Section IV.

B. Simulated Aircraft Dynamics

The adaptive critic controller is implemented on a nonlinear six degrees-of-freedom (6-DOF) simulation of a business jet aircraft called FLIGHT [35]. The simulated equations of motion (20) are based on mathematical models, full-scale wind tunnel data, and characteristics of an early twin-jet configuration. Spherical–trigonometric relationships [41] are used to compute the set point values of the roll angle and angle of attack for the large-angle maneuvers simulated in this paper.

The longitudinal state vector includes the velocity V, the path angle γ, the pitch rate q, and the pitch angle θ, i.e., $\mathbf{x}_L = [V\ \gamma\ q\ \theta]^T$. The lateral–directional state vector includes the yaw rate r, the sideslip angle β, the roll rate p, and the bank angle μ, i.e., $\mathbf{x}_{LD} = [r\ \beta\ p\ \mu]^T$ and $\mathbf{x} \equiv \left[\mathbf{x}_L^T\ \mathbf{x}_{LD}^T\right]^T$. The scheduling vector contains dynamically significant variables that are the auxiliary inputs of the neural network $\mathbf{a} = [V\ H]^T$. The longitudinal control vector consists of the throttle δT and stabilator δS, or $\mathbf{u}_L = [\delta T\ \delta S]^T$. The lateral–directional control vector consists of the aileron δA, and the rudder δR, or $\mathbf{u}_{LD} = [\delta A\ \delta R]^T$ and $\mathbf{u} \equiv \left[\mathbf{u}_L^T\ \mathbf{u}_{LD}^T\right]^T$. The actual values of $\mathbf{x}(t_\kappa)$ and $\mathbf{a}(t_\kappa)$ are observed from the simulated aircraft every $\Delta t = 0.1$ s. The neural network controller is then updated at every time step using the corresponding output sample $\mathbf{u}(t_\kappa)$, computed by the adaptive critic architecture. A set of $q = 14$ steady-level flight conditions or *design points*, illustrated in Fig. 3, is chosen to design the set of classical control gains.

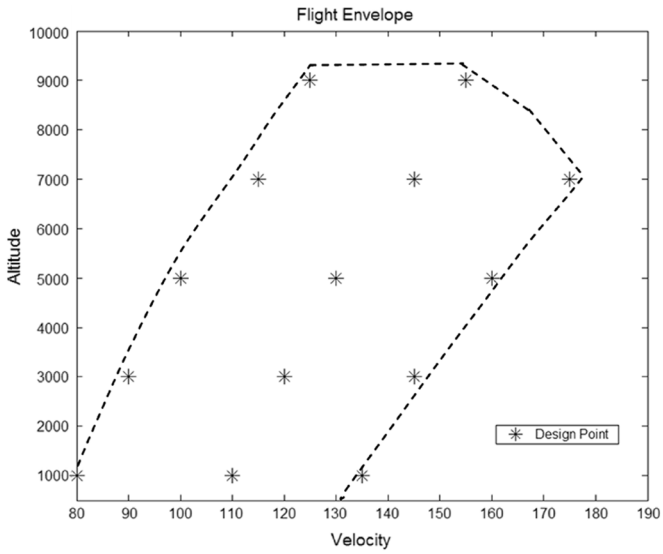

Fig. 3. Aircraft steady-level flight envelope (or LPV regime) is plotted together with the set of 14 equilibria $\mathcal{P}$, used to derive the classical control gains that comprise the LTM of the neural network controller.

C. Motivation for Constrained Training in Adaptive Critic Flight Control

The neural network flight controller reviewed in Section V-A was shown to adapt online to a variety of maneuvers and conditions, such as nonlinear maneuvers, parameter variations, and multiple control failures, without any prior knowledge of the upcoming conditions [17]. The adaptation was so effective as to prevent loss of control, as illustrated by the maneuver in Fig. 4, taken from [17]. In this figure, a -70° turn is commanded at an airspeed of 160 m/s and altitude of 7000 m. At this angle, the aircraft cannot generate sufficient lift to maintain altitude, and the coupled dynamics become so significant that they compromise the classical gain-scheduled controller's performance, leading to loss of aircraft control. Although not typical, these conditions could result from an emergency situation and require the pilot to take control of the aircraft, because with such a large bank angle it is difficult to coordinate the turn [35], [41]. Also, a fixed-parameter control system tends to require unreasonable throttle usage. Instead, the adaptive neural controller learns from the nonlinear aircraft dynamics as well as from the

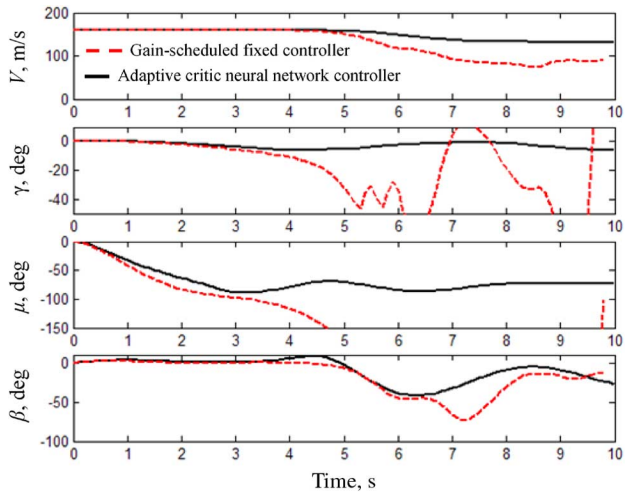

Fig. 4. Comparison between adaptive critic neural controller (solid line) and fixed-parameter gain-scheduled controller (dashed line) subject to a -70° roll angle step command at $(V_0, H_0) = (160$ m/s, 7 km), taken from [17].

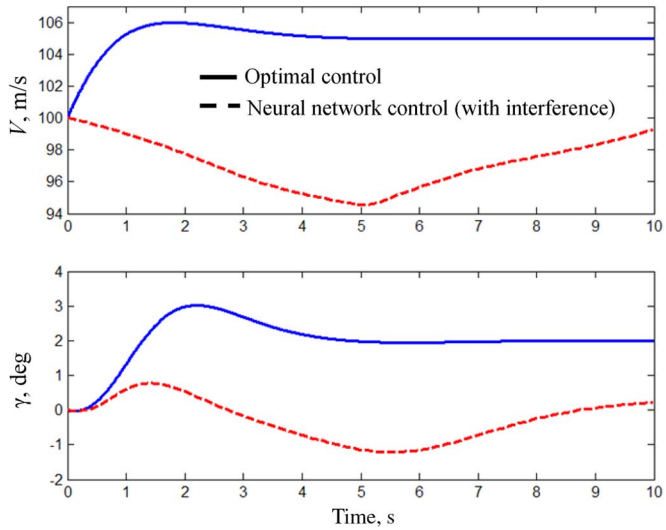

Fig. 5. Loss of LTM is illustrated by comparing the performance of the neural controller (dashed line) with parameters updated during the -70° turn (Fig. 7) and held fixed, to the known optimal response (solid line) for a small-angle maneuver with $\Delta\gamma = 2^\circ$ and $\Delta V = 5$ m/s, at the design point $(V_0, H_0) =$ (100 m/s, 5000 m).

control bounds [17], finally preventing loss of stability and coordinating the turn.

The adaptive neural network controller was found to preserve some memory of its performance baseline (or $\mathcal{T}_{\rm LTM}$) during most of the individual maneuvers described in [17]. However, further research showed that this memory can be compromised when several nonlinear maneuvers are presented before returning to the steady-level flight envelope. Also, the LTM can deteriorate during difficult maneuvers, such as the one illustrated in Fig. 4, which requires the adaptive controller to learn from dynamics that are significantly nonlinear. To illustrate this point, in Fig. 5, the neural network controller implementing the parameters learned during the -70° turn (and held fixed) is used to control a small-angle maneuver at one of the design points. Under these conditions, the classical controller (23) is known to perform optimally and its response

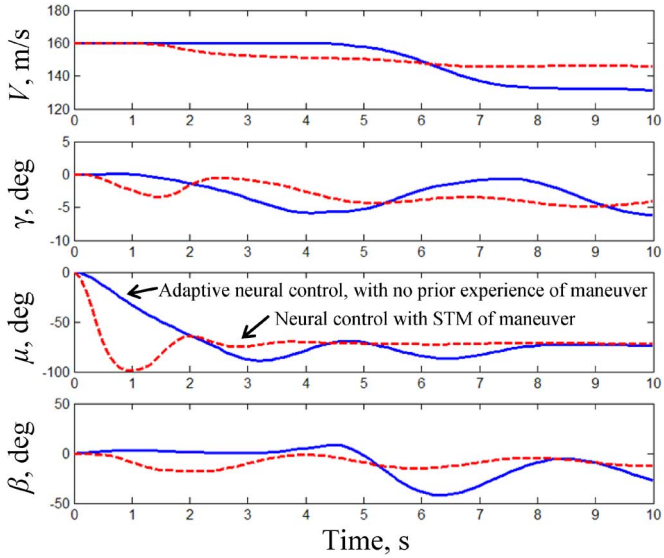

Fig. 6. Value of the newly acquired STM is illustrated by the improved performance of the neural controller with the new parameters (updated during the -70° turn in Fig. 4 and held fixed), shown in a dashed line, compared to when this maneuver was experienced for the first time (solid line), at $(V_0, H_0) = (160$ m/s, 7000 m).

is plotted for comparison (solid line). Prior to adaptation, the neural network controller would have performed optimally as well. However, its new performance (dashed line in Fig. 5) is considerably degraded with respect to the optimal baseline. This result is representative of other simulations performed throughout $\mathcal{X}_{\rm LPV}$, and illustrates that memory of the classical gain-scheduled design (or $\mathcal{T}_{\rm LTM}$) has been degraded due to interference. As a consequence, the neural controller would need to relearn how to control the aircraft everywhere in $\mathcal{X}_{\rm LPV}$ through the DHP architecture, even though the gain-scheduled designs are known *a priori* and have been embedded in the neural controller offline.

On the other hand, the newly acquired control knowledge (or $\mathcal{T}_{\rm STM}$) is effectively utilized by the neural controller in the short term. Suppose the same neural controller (with parameters updated during the -70° turn in Fig. 4, and held fixed) is used to control another -70° turn, as shown in Fig. 6. Even without undergoing additional adaptation, the controller's performance is much improved with respect to the first time this maneuver was performed (solid line). This can be seen by the reduced oscillations and settling time of the response plotted as a dashed line in Fig. 6. If the adaptation is allowed to continue, the neural controller continues to optimize its performance until the optimality condition is satisfied within a desired tolerance. These results, together with [17], confirm that the control knowledge acquired online through DHP is useful when new dynamics are experienced for the first time, or in the short term. However, repeated incremental training sessions can potentially alter the controller's performance in $\mathcal{X}_{\rm LPV}$.

D. Constrained Training Implementation and Results

In this section, we demonstrate that, by implementing the constrained training approach, the LTM of the gain-scheduled controllers can be retained at all times during the adaptation

TABLE I
COMPARISON OF THE MSE OVER THE LTM TRAINING SET

Training Algorithm	Neural Network Controller MSE	$t_\kappa = 0$ (s)	$t_\kappa = 5$ (s)	$t_\kappa = 10$ (s)
Unconstrained RPROP	Output (LTM) data	$2.729 \cdot 10^{-7}$	$1.770 \cdot 10^{12}$	$3.168 \cdot 10^{11}$
	Gradient (LTM) data	$8.470 \cdot 10^{-28}$	$7.868 \cdot 10^{-4}$	$1.373 \cdot 10^{-4}$
Constrained RPROP	Output (LTM) data	$2.729 \cdot 10^{-7}$	$2.404 \cdot 10^{-7}$	$5.555 \cdot 10^{-7}$
	Gradient (LTM) data	$8.470 \cdot 10^{-28}$	$7.545 \cdot 10^{-26}$	$4.057 \cdot 10^{-27}$

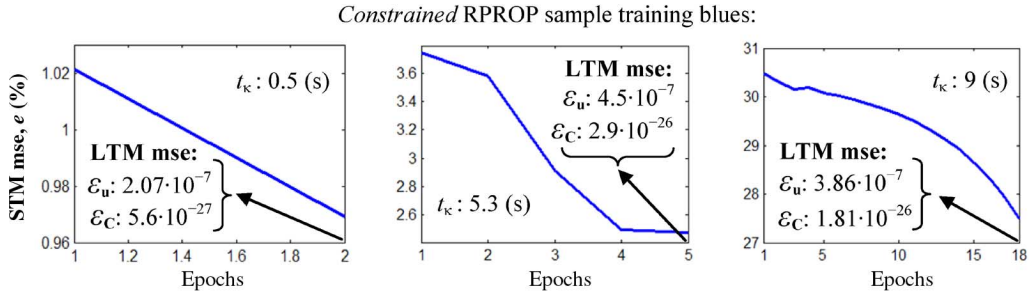

Fig. 7. Three constrained training sessions are illustrated during which the STM error e is minimized, and the LTM constraints are satisfied within very small MSEs denoted by $\varepsilon_{\mathbf{u}}$ and $\varepsilon_{\mathbf{C}}$.

while new STMs are formed. LTM preservation is verified by comparing the mean squared error (MSE) of the neural network controller over $\mathcal{T}_{\text{LTM}}$ before and after the adaptation. Numerical results show that this error remains close to zero at all times. Also, LTM effectiveness is illustrated by testing the performance of the neural network controller throughout the steady-level flight envelope (Fig. 3) after the adaptation has taken place.

Classical gain-scheduled controllers (Section V-A1) are used to form a gradient-based LTM training set $\mathcal{T}_{\text{LTM}} = \{[\mathbf{0}\ (\mathbf{a}^k)^T]^T, \mathbf{0}, \mathbf{C}_L^k, \mathbf{C}_{LD}^k\}_{k=1,\ldots,14}$. Using the results in Section IV, this set can be embedded in the equality constraints

$$\begin{cases} \mathbf{0}_{m\times q} = \mathbf{V}\boldsymbol{\Sigma} + \mathbf{B} \\ \mathbf{C}_L^k = \mathbf{V}(\Theta_1, H)\boldsymbol{\Lambda}^k \mathbf{W}_{\mathbf{x}}(H, I_1) \\ \mathbf{C}_{LD}^k = \mathbf{V}(\Theta_2, H)\boldsymbol{\Lambda}^k \mathbf{W}_{\mathbf{x}}(H, I_2) \end{cases}, \qquad k = 1, \ldots, 14 \tag{26}$$

for which all variables are defined in Section IV and the adjoined gradient is provided in Theorem 2. The neural network controller (9) has $s = 2q = 28$ nodes, and is trained with $\mathcal{T}_{\text{LTM}}$ prior to online implementation. If its performance becomes suboptimal, the optimality condition (25) produces a target $\mathbf{u}(t_\kappa)$ at every time step t_κ, which is immediately used by Algorithm 1 to update the weights of (9).

To initiate the DHP adaptation, the aircraft is directed to execute a large-angle turn with $\Delta\beta = 60^\circ$ at $(V_0, H_0) =$ (145 m/s, 6000 m). As was the case in the example provided in Fig. 4, this maneuver also violates both assumptions of small deviations and decoupled dynamics. Consequently, a classical controller (23) designed specifically for such flight conditions is suboptimal, as can be verified by evaluating the optimality condition (25). For comparison, the neural network controller is updated both with an unconstrained RPROP algorithm [17] and with a constrained RPROP algorithm implementing the adjoined gradient. The constrained adaptive neural network controller is found to outperform both the linear design and the unconstrained adaptive controller, leading to an overall reduction in total cost of up to 62.5% by the final time $t_f = 10$ s (compared to the other controllers). Also, by the end of the maneuver, the constrained-adaptive neural network's performance is found to be nearly optimal, as shown by the incremental cost $\mathcal{L}_D \to 0$.

At the same time, by implementing the adjoined gradient, the constrained neural network controller retains accurate memory of the LTM training set $\mathcal{T}_{\text{LTM}}$, as shown in Table I, which summarizes the MSE over $\mathcal{T}_{\text{LTM}}$ tested at sample time steps during the adaptation. It can be seen that throughout the adaptation the neural network controller trained with the constrained optimization algorithm maintains an MSE of $\mathcal{O}(10^{-7})$ for the output data, and an MSE lower than $\mathcal{O}(10^{-26})$ for the gradient data. The same DHP controller trained with an unconstrained RPROP algorithm considerably worsens the performance over the LTM training set, increasing the MSE by up to 24 orders of magnitude shortly after the adaptation begins. The training blues obtained from sample training sessions conducted with constrained and unconstrained RPROP algorithms are shown in Figs. 7 and 8, respectively. Each sample training session occurs at the time instant t_κ indicated on the plot. In these training blues, the STM error e, defined in (5), is plotted with respect to the epochs used for its minimization. The LTM MSEs for the output and gradient data, denoted by $\varepsilon_{\mathbf{u}}$ and $\varepsilon_{\mathbf{C}}$, respectively, are computed at the end of each training session using the

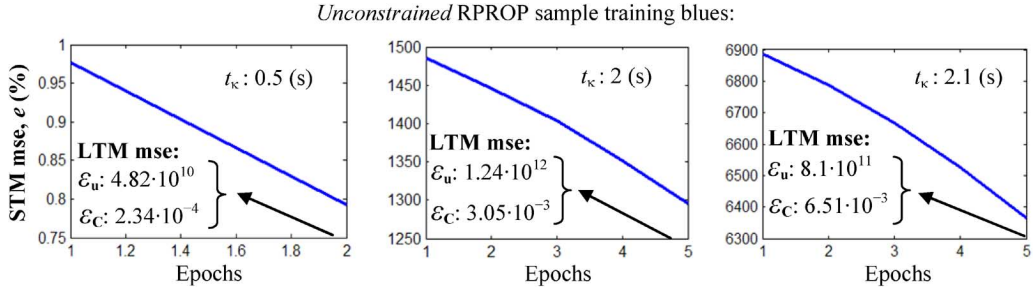

Fig. 8. Three unconstrained training sessions are illustrated during which the STM error e is minimized, but the LTM constraints are not satisfied, as shown by the large MSEs $\varepsilon_\mathbf{u}$ and $\varepsilon_\mathbf{C}$.

TABLE II
PERCENT CHANGE IN ACTION NETWORK WEIGHTS UPDATED BY DHP ROUTINE

Training Algorithm	**Percent Change:**	Minimum (%)	Mean (%)	Maximum (%)
Unconstrained RPROP	Input weights, $\mathbf{W_x}$	$5.2 \cdot 10^{-4}$	87.3	$1.8 \cdot 10^{4}$
	Output weights, $\mathbf{V}$	$5.3 \cdot 10^{-4}$	74.3	100
Constrained RPROP	Input weights, $\mathbf{W_x}$	$7.9 \cdot 10^{-3}$	68.7	903.7
	Output weights, $\mathbf{V}$	0.007	79.3	708.5

updated parameters, and shown in Figs. 7 and 8. Similar LTM MSE values are obtained at intermediate epochs, and in all other training sessions.

A previous approach to suppressing interference without representing the LTM data proposed to hold the LTM weights constant in an attempt to preserve the corresponding memories [4], [5]. However, due to the nonlinear nature of neural networks, when the STM weights are modified, the old values of the LTM weights may no longer match the LTM data, or the constraints (3). From (4), it is clear that the values of $\mathbf{L}$ that satisfy (3) depend on the values of $\mathbf{S}$. Our simulations confirm that, by holding the LTM weights of the neural network controller constant during the adaptation, its performance over the steady-level envelope deteriorates with rapid increase of the LTM MSE as the STM weights are updated to minimize e.

In our constrained optimization approach, all weights are allowed to change during incremental training, but the LTM weights change to preserve the $\mathcal{T}_{\mathrm{LTM}}$ memory, while the STM weights change to acquire new memories from $\mathcal{T}_{\mathrm{STM}}$. Table II shows that the mean, minimum, and maximum percent changes in the weights' magnitudes produced by the constrained RPROP algorithm are comparable to those produced by the unconstrained RPROP algorithm, which updates all of the neural network weights, without distinction, for the only purpose of minimizing e over $\mathcal{T}_{\mathrm{STM}}$. In some instances, the percent changes are larger for the constrained RPROP algorithm, because it must also satisfy the LTM constraints.

Finally, the effectiveness of the LTM is demonstrated by testing the updated neural network controller in the steady-level envelope. After the adaptation has taken place, the neural network controller is purposely applied to a small-angle maneuver for which the optimal solution is known from classical control (Section V-A1) and is plotted in Figs. 9 and 10 in a solid line for comparison. As shown in Section V-C (Fig. 5), even if knowledge of the classical gain-scheduled controllers is initially embedded in the neural network controller, over time it can be deteriorated by the adaptation due to interference. Instead, when the adaptation is conducted by means of a constrained RPROP algorithm, the steady-level response of the updated neural network controller (dotted line) exactly overlaps that of the optimal control law (solid line), as shown in Fig. 9, and so does its cost (Fig. 10). Also for comparison, an otherwise identical adaptive critic controller is updated using an unconstrained RPROP training algorithm during the same large-angle maneuver. Although both controllers were trained offline with the same gain-scheduled controllers, the unconstrained neural network performance in the steady-level regime is now suboptimal, as can be seen from the response to a small-angle maneuver and the cost plotted in Figs. 9 and 10 (dashed line). By preserving all of its LTM virtually intact (Table I), the constrained adaptive neural network controller maintains the desired baseline performance, and is capable of assimilating new memories through the same well-known adaptive critic architecture.

VI. CONCLUSION

A novel approach has been developed for training neural networks with global support by means of a constrained optimization framework that encodes LTM in the form of equality constraints. The solution of the resulting constrained-optimization problem is simplified by introducing an adjoined error gradient that eliminates the need for direct substitution of the constraint equation in the network error function, and exploits the error derivatives computed by classical backpropagation.

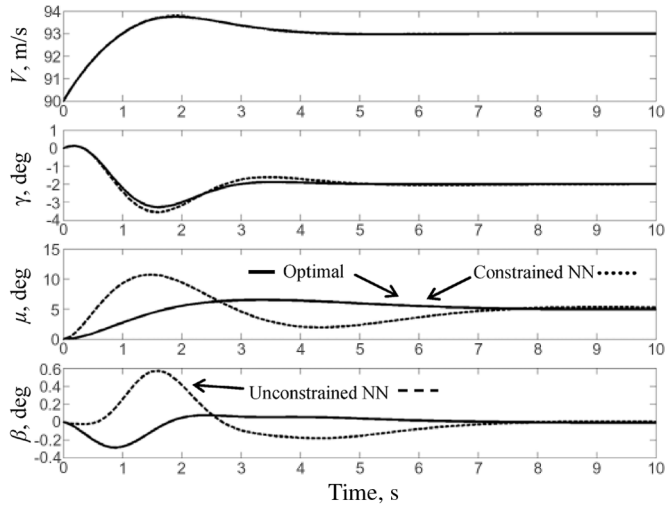

Fig. 9. Small-angle command is imparted ($\Delta V = 3$ m/s, $\Delta\gamma = -2°$, and $\Delta\beta = 5°$), at $(V_0, H_0) = (90$ m/s, 3000 m), in order to illustrate that even after adaptation the constrained adaptive neural network controller (dotted line) performs identically to the optimal PI controller (solid line) designed specifically for these conditions, while the adaptive neural network controller updated with an unconstrained RPROP (dashed line) is now suboptimal due to interference.

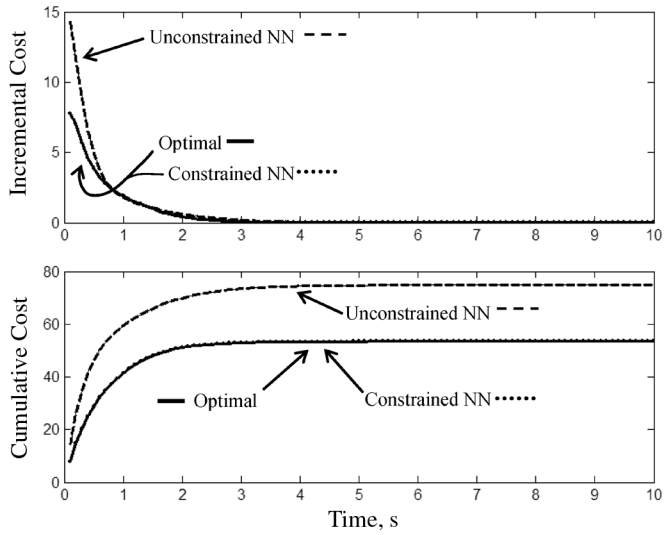

Fig. 10. Incremental and cumulative costs for the maneuver in Fig. 9 confirm that while the constrained adaptive controller's performance is optimal (the dotted and solid lines overlap in these graphs), the controller updated by the unconstrained RPROP algorithm performs suboptimally after the adaptation, and thus its memory of the optimal PI controller has been deteriorated.

The approach is applied to a well-known adaptive critic flight controller that is trained incrementally to adapt to new flight regimes online. Meanwhile, it must also maintain well-established control knowledge (or LTM) over the steady-level flight envelope, where the aircraft operates most of the time. Prior research has shown that a traditional, unconstrained training algorithm (such as RPROP) can deteriorate prior control knowledge shortly after the onset of the adaptation, due to interference. By implementing a constrained-training RPROP algorithm, the LTM is preserved with very high accuracy at all times, while still allowing the STM network error to be minimized, as required by the DHP architecture.

Appendix I
Implicit Function Theorem

A. Theorem 3 (Implicit Function)

Consider a system of n equations in $m+n$ variables

$$f(\mathbf{y}, \mathbf{x}) = 0 \tag{27}$$

where $f : \mathbf{R}^{m+n} \to \mathbf{R}^n$, $\mathbf{x} \in \mathbf{R}^m$, and $\mathbf{y} \in \mathbf{R}^n$. Let $\mathcal{S}$ be an open subset of $\mathbf{R}^{m+n}$, and $f : S \to \mathbf{R}^n$ be a function such that for some $p \geq 0$, $f \in C^p$ over $\mathcal{S}$, and assume that $(\partial f/\partial \mathbf{y})|_{(\mathbf{y},\mathbf{x})}$ exists and is continuous in $\mathcal{S}$. Let $(\bar{\mathbf{y}}, \bar{\mathbf{x}}) \in \mathcal{S}$ be a vector such that $f(\bar{\mathbf{y}}, \bar{\mathbf{x}}) = 0$ and the matrix $(\partial f/\partial \mathbf{y})|_{(\bar{\mathbf{y}},\bar{\mathbf{x}})}$ is nonsingular. Then, (27) uniquely specifies a function $\phi \in C^p$ over a neighborhood $\mathcal{N}$ of $\bar{\mathbf{x}}$ such that $\bar{\mathbf{y}} = \phi(\bar{\mathbf{x}})$, and $f[\mathbf{x}, \phi(\mathbf{x})] = 0$ for all $\mathbf{x}$ in $\mathcal{N}$.

This implicit function theorem [29, pp. 11–12] can be applied to the matrix equality constraint (3) using standard methods for matrix functions and derivatives [42, pp. 408–411]. Let $\mathbf{y} \equiv \text{vec}(\mathbf{L})$ and $\mathbf{x} \equiv \text{vec}(\mathbf{S})$, where the vec operator obtains an $(rs \times 1)$ vector by stacking the columns of an $(r \times s)$ matrix [42]. Then, (3) can be written as the system in (27), where n and m represent the number of LTM and STM weights, respectively. Now, let the matrix function $\mathcal{C} \equiv \text{vec}^{-1}(\phi)$. Because $\mathbf{L} = \text{vec}^{-1}[\text{vec}(\mathbf{L})]$, then if (27) uniquely specifies the function $\phi \in C^p$ with the aforementioned properties, it also follows that $g[\mathcal{C}(\mathbf{S}), \mathbf{S}] = 0$, with $\mathbf{S} = \text{vec}^{-1}(\mathbf{x})$ for all $\mathbf{x}$ in $\mathcal{N}$.

Appendix II
Properties of the Adjoined Error Gradient

Training a neural network by a constrained-optimization approach requires computing the adjoined error gradient with respect to the weights at every epoch. In this section, we present a set of properties to simplify the analytical calculation of this gradient, and to formulate it in terms of derivatives that are computed by classical backpropagation. Additional properties of derivatives and chain rule for matrix functions can be found in [43] and [44]. Tensor notation is used for simplicity.

A. Adjoined Gradient Property 1 (Recursion)

The adjoined gradient can be applied recursively to subsequent equality constraints on the neural network weights. Suppose that

$$E(\mathbf{S}) = e\left(\mathcal{C}(\mathbf{S}), \mathbf{S}\right) \quad \text{subject to} \tag{28}$$

$$\mathbf{L} = \mathcal{C}(\mathbf{S}) \quad \text{and} \quad \mathbf{S} = \mathcal{F}(\mathbf{M}) \tag{29}$$

where $\mathcal{C}$ and $\mathcal{F}$ denote two sets of algebraic equations, and $\mathbf{M}$ denotes any subsets of LTM weights. Then

$$\partial_{\mathbf{S}} E = \partial_{\mathbf{S}} e + \mathcal{G}[\mathbf{L}, \mathbf{S}, \partial_{\mathbf{L}} e] \tag{30}$$

$$\partial_{\mathbf{M}} E = \mathcal{G}[\mathbf{S}, \mathbf{M}, \partial_{\mathbf{S}} E] = \mathcal{G}\left[\mathbf{S}, \mathbf{M}, \partial_{\mathbf{S}} e + \mathcal{G}[\mathbf{L}, \mathbf{S}, \partial_{\mathbf{L}} e]\right] \tag{31}$$

where, in this case, $\mathbf{M}$ is not an argument of e. Thus, $\partial_{\mathbf{M}} e = 0$.

B. Adjoined Gradient Property 2 (Linear Transformation)

Let $\mathbf{S}$ be given by the following matrix multiplication:

$$\mathbf{S} = \mathbf{K} \cdot \mathbf{M}. \tag{32}$$

Because $s_{ij} = k_{ir}m_{rj}$, the chain rule (8) yields $\partial E/\partial m_{rj} = k_{ir}\partial E/\partial s_{ij}$. Therefore, the adjoined gradient with respect to the new matrices is given by

$$\partial_{\mathbf{M}} E = \mathbf{K}^T \cdot \partial_{\mathbf{S}} E \tag{33}$$

and

$$\partial_{\mathbf{K}} E = \partial_{\mathbf{S}} E \cdot \mathbf{M}^T. \tag{34}$$

This property can be easily extended to repeated matrix multiplications. For example, if

$$\mathbf{S} = \mathbf{M} \cdot \mathbf{M} = \mathbf{M}^2 \tag{35}$$

then the gradient with respect to the matrix $\mathbf{M}$ is given by

$$\partial_{\mathbf{M}} E = \partial_{\mathbf{S}} E \cdot \mathbf{M}^T + \mathbf{M}^T \cdot \partial_{\mathbf{S}} E. \tag{36}$$

C. Adjoined Gradient Property 3 (Inverse of a Matrix)

Let the matrix $\mathbf{M}$ be defined in terms of an invertible matrix $\mathbf{S}$

$$\mathbf{M} = \mathbf{S}^{-1} \tag{37}$$

and assume the adjoined gradient is known with respect to $\mathbf{S}$. We seek to compute the adjoined gradient $\partial_{\mathbf{M}} E$ in terms of $\partial_{\mathbf{S}} E$. The inverse of a matrix can be computed from its adjoint and determinant

$$\mathbf{S} = \mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})}\mathrm{adj}(\mathbf{M}) \tag{38}$$

or

$$\mathbf{M}^{-1} = \{s_{ij}\} = \left\{\frac{M_{ji}}{\det(\mathbf{M})}\right\} \tag{39}$$

where M_{ji} is the *cofactor* of the i,j element in $\mathbf{M}$. The cofactor of a matrix element is obtained from its corresponding *minor*, using the following relationship:

$$M_{ij} = (-1)^{i+j}\det(\mathbf{K}_{ij}) \tag{40}$$

where the minor of the cofactor M_{ij} is the determinant of a matrix $\mathbf{K}_{ij}$, obtained by removing the row and column containing the i,j element of $\mathbf{M}$, from $\mathbf{M}$. The adjoined gradient with respect to the new variable is obtained from the chain rule

$$\partial_{\mathbf{M}} E = \partial_{\mathbf{S}} E \cdot \partial_{\mathbf{M}} \mathbf{S} \tag{41}$$

where

$$\begin{aligned}\partial_{\mathbf{M}}\mathbf{S} &\equiv \left\{\frac{\partial s_{ij}}{\partial m_{kl}}\right\} \\ &= (-1)^{i+j}\frac{1}{\det^2(\mathbf{M})}\left[\frac{\partial\left[\det(\mathbf{K}_{ji})\right]}{\partial m_{kl}}\cdot\det(\mathbf{M})\right. \\ &\qquad \left. - \frac{\partial\left[\det(\mathbf{M})\right]}{\partial m_{kl}}\cdot\det(\mathbf{K}_{ji})\right].\end{aligned} \tag{42}$$

In order to determine the partial derivatives of the previous expression, consider the cofactor expansion of the determinant of a matrix in tensor notation, i.e., $\det(\mathbf{M}) = m_{kr}M_{kl}\delta_{rl} = m_{kl}M_{jl}\delta_{kj}$, where δ_{ij} is the Kronecker delta function. Together with (40), this expansion can be used to obtain the following simplifying relationships, namely:

$$\frac{\partial\left[\det(\mathbf{M})\right]}{\partial m_{kl}} = M_{kl} \tag{43}$$

$$\frac{\partial\left[\det(\mathbf{K}_{ji})\right]}{\partial m_{kl}} = N_{rt}(1-\delta_{ri})(1-\delta_{lj}) \tag{44}$$

with

$$r = \begin{cases} k, & \text{if } k \le i \\ (k-1), & \text{if } k > i \end{cases} \quad \text{and} \quad t = \begin{cases} l, & \text{if } l < j \\ (l-1), & \text{if } l > j \end{cases} \tag{45}$$

and where $\mathbf{N}$ is the cofactor of $\mathbf{K}_{ij}$. Equations (44) and (45) can be written in a more compact notation by introducing a matrix $\mathbf{N}'_{ij}$ that is obtained from $\mathbf{K}_{ij}$ by substituting the row and column containing the i,j element of the matrix $\mathrm{adj}(\mathbf{K}_{ij})^T$ with a row and column of zero elements. Then, (42) can be expressed as

$$\begin{aligned}\partial_{\mathbf{M}} E &\equiv \left\{\frac{\partial E}{\partial m_{kl}}\right\} \\ &= (-1)^{i+j}\left[\frac{\partial E}{\partial s_{ij}}\left(\frac{\det(\mathbf{M})\mathbf{N}'_{ij} - \det(\mathbf{K}_{ij})\mathrm{adj}(\mathbf{M})^T}{\det^2(\mathbf{M})}\right)\right].\end{aligned} \tag{46}$$

Provided that $\mathbf{K}_{ij}$ is invertible, one can also define a matrix $\mathbf{K}'_{ij}$ that is obtained from $\mathbf{K}_{ij}^{-1}$ by substituting its ith row and jth column with zero elements. Then, the adjoined gradient of the inverse matrix $\partial_{\mathbf{M}} E$, in terms of $\partial_{\mathbf{S}} E$, can be further simplified to

$$\partial_{\mathbf{M}} E = s_{ij}\frac{\partial E}{\partial s_{ij}}\left[\mathbf{K}'_{ij} - \mathbf{S}\right]^T \tag{47}$$

by summing over both i and j according to tensor rules. If $\mathbf{K}_{ij}$ is not invertible, the use of (46) is required. It can be seen from the previous equations that even the most efficient computation of the adjoined gradient with respect to an $(n\times n)$ inverse matrix is expensive, as it requires n^2 inversions of an $(n-1)\times(n-1)$ matrix, $\mathbf{K}'_{ij}$.

D. Adjoined Gradient Property 4 (Kronecker Product)

Suppose that $\mathbf{S}$ is obtained from the Kronecker product $(\otimes)$ of two matrices

$$\mathbf{S} = \mathbf{M} \otimes \mathbf{K} \tag{48}$$

and that $\partial_{\mathbf{S}} E$ is given. Then, the adjoined gradient with respect to $\mathbf{M}$ can be computed as

$$\partial_{\mathbf{M}} E = \partial_{\mathbf{S}} E \otimes \mathbf{K}. \tag{49}$$

Finally, for any smooth and differentiable nonlinear function $f : \mathcal{R} \to \mathcal{R}$, if the adjoined gradient is known with respect to the matrix

$$\mathbf{S} = \{s_{ij}\} = f(m_{ij}) \tag{50}$$

then the adjoined gradient with respect to $\mathbf{M} = \{m_{ij}\}$ is given by

$$\partial_{\mathbf{M}} E = \partial_{\mathbf{S}} E \otimes \left\{ \frac{\partial f}{\partial m_{ij}} \right\}. \tag{51}$$

Appendix III
Positional Notation

Positional notation is used to represent neural network weights based on the inputs, hidden nodes, and outputs they connect. Any vector can be viewed as an ordered set of elements. Let I denote the index set of the neural input vector $\mathbf{x}$, Θ the index set of the output vector $\mathbf{z}$, and H the index set of the hidden nodes $\Phi(\mathbf{n})$, and of the input-to-node vector $\mathbf{n}$. Suppose the input vector is partitioned into two or more vectors $\mathbf{x} = [\mathbf{x}_1^T \dots \mathbf{x}_m^T]^T$, then each vector partition $\mathbf{x}_j$ is a subset of $\mathbf{x}$, with an index set denoted by I_j. Similarly, the subscript of the hidden-node and output index sets denote the corresponding vector partitions. Therefore, the third-order tensor $\mathbf{U}(H_l, I_j, \Theta_i)$ contains the neural network weights associated with the hidden nodes, inputs, and outputs with index sets H_l, I_j, and Θ_i, respectively. The same notation is used for weight matrices (or second-order tensors), using two arguments instead of three. For example, consider the neural network described in Fig. 1, with input weights $\mathbf{W}$. The matrix $\mathbf{W}(H_2, I_1)$ denotes the input weights that connect the inputs in $\mathbf{x}_1$ to the hidden nodes $\Phi(\mathbf{n}_2)$. Consequently, the matrix $\mathbf{W}(H_2, I_1)$ is easily obtained by removing the rows and columns in $\mathbf{W}$ with index sets $\bar{H}_2$ and $\bar{I}_1$ [where $\bar{(\cdot)}$ denotes the complement set].

Appendix IV
Proof of Theorem 1 (Equality Constraints)

The equality constraints (12) are derived by considering the neural network equation (9) and its derivatives

$$\frac{\partial z_i}{\partial p_j} = v_{il} \frac{\partial \sigma}{\partial n}(n_l) w_{lj} \equiv v_{il} \sigma'(n_l) w_{lj}. \tag{52}$$

The neural network hidden nodes are partitioned as follows:

$$\mathbf{n} \equiv \{n_i\} = \{w_{ij} p_j\} = \begin{bmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \end{bmatrix} \equiv \{\mathbf{n}_l\}, \qquad l = 1, 2. \tag{53}$$

Similarly, the neural network output can be partitioned into the subvectors $\mathbf{z} = \{\mathbf{z}_i\}$, such that $\mathbf{z}_i$ and $\mathbf{u}_i$ have the same index set. Let $\mathbf{Z}$, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{\Lambda}^k$ be defined as in Theorem 1, and let $\mathbf{W} = [\mathbf{W_x}\ \mathbf{W_a}]$. Then, adopting the notation in Appendix III, the neural network (9) and its derivatives (52) match the training set $\mathcal{T}_{\mathrm{LTM}}$ when the weights satisfy the following algebraic equations:

$$\begin{cases} \mathbf{Z} = \mathbf{V}\Phi(\mathbf{W_a A}) + \mathbf{B} \equiv \mathbf{V\Sigma} + \mathbf{B} \\ \mathbf{C}_{ij}^k = \mathbf{V}(\Theta_i, H)\mathbf{\Lambda}^k \mathbf{W_x}(H, I_j), \\ \quad (i,j) = (1,1), (2,2), \qquad k = 1, \dots, q. \end{cases} \tag{54}$$

It can be seen from (54) that if $s = 2q$ and $\mathbf{W_a}$ is known, both systems are linear and square, and yield an exact solution provided they are full rank (as in [30]).

Now, we can partition $\mathbf{Z}$ into two matrices $\mathbf{Z}_i$ $(i = 1, 2)$, such that $\mathbf{Z}_i \equiv [\mathbf{z}_i^1 \cdots \mathbf{z}_i^q]$. The matrix $\mathbf{\Sigma}$ is obtained from Φ operating on the LTM training set inputs, for which $\mathbf{x}^k = \mathbf{0}$ for $\forall k$, thus $\mathbf{\Sigma} \equiv \Phi(\mathbf{W_a A})$. The positional notation can be used to refer to the submatrices $\mathbf{\Sigma}(H_l) \equiv \Phi[\mathbf{W_a}(H_l, I) \cdot \mathbf{A}]$ and $\mathbf{\Lambda}^k(H_l) \equiv \mathrm{diag}[\Phi'(\mathbf{n}_l^k)]$. Hence, (54) can be written as (12), emphasizing the input, hidden-node, and output partitions.

Appendix V
Proof of Proposition 1 (Explicit Constraints)

Let $\mathbf{K}_{ij}$ and $\mathbf{\Lambda}$ be defined as in (18). The positional notation can be used to refer to the submatrix $\mathbf{\Lambda}(H_l) \equiv [(\mathbf{\Lambda}^1(H_l))^T \cdots (\mathbf{\Lambda}^q(H_l))^T]^T$. Then, through simple matrix manipulations, the second constraint in (12), consisting of q systems of equations, can be rewritten as

$$\begin{aligned} \mathbf{K}_{ij} = \mathrm{diag}[\mathbf{V}(\Theta_i, H_1)]\mathbf{\Lambda}(H_1)\mathbf{W_x}(H_1, I_j) + \mathrm{diag}[\mathbf{V}(\Theta_i, H_2)] \\ \cdot \mathbf{\Lambda}(H_2)\mathbf{W_x}(H_2, I_j), \quad (i,j) = (1,1), (2,2) \end{aligned} \tag{55}$$

where $\mathbf{V}_d(\Theta_i, H_l) \equiv \mathrm{diag}[\mathbf{V}(\Theta_i, H_l)]$ denotes a block-diagonal matrix obtained by placing q identical matrices $\mathbf{V}(\Theta_i, H_l)$ on the diagonal of a zero matrix of appropriate dimensions. Then, (12) can be written as

$$\begin{cases} \mathbf{V}(\Theta_i, H_l) = -\left[\mathbf{B}(\Theta_i) - \mathbf{Z}_i + \mathbf{V}(\Theta_i, \bar{H}_l)\mathbf{\Sigma}(\bar{H}_l)\right] \\ \quad \cdot \mathbf{\Sigma}(H_l)^{-1} \\ \mathbf{W_x}(H_l, I_j) = [\mathbf{V}_d(\Theta_i, H_l)\mathbf{\Lambda}(H_l)]^{-1} \\ \quad \cdot \left[\mathbf{K}_{ij} - \mathbf{V}_d(\Theta_i, \bar{H}_l)\mathbf{\Lambda}(\bar{H}_l)\mathbf{W_x}(\bar{H}_l, I_j)\right] \end{cases} \tag{56}$$

with $i = j = l$. Because $i, j, l = 1, 2$, it follows that $\bar{H}_2 = H_1$ and $\bar{H}_1 = H_2$. Finally, substituting $i = j = l$ with $\imath = 1, 2$, and using the definitions in Proposition 1, (56) can be written as (17).

Appendix VI
Proof of Theorem 2 (Adjoined Gradient)

Using the *recursion* and the *linear transformation* properties (Appendix II), the adjoined gradient with respect to the STM input weights is computed by differentiating the corresponding constraint in (17)

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{W_x}(\bar{H}_\imath, I_\imath)} &= \frac{\partial E}{\partial \mathbf{W_x}}(\bar{H}_\imath, I_\imath) \\ &= \frac{\partial e}{\partial \mathbf{W_x}}(\bar{H}_\imath, I_\imath) + \\ &\quad - \left\{ [\mathbf{V}_d(\Theta_\imath, H_\imath)\mathbf{\Lambda}(H_\imath)]^{-1} \right. \\ &\quad \left. \cdot \left[\mathbf{V}_d(\Theta_\imath, \bar{H}_\imath)\mathbf{\Lambda}(\bar{H}_\imath)\right] \right\}^T \\ &\quad \cdot \frac{\partial e}{\partial \mathbf{W_x}}(H_\imath, I_\imath), \qquad \imath = 1, 2 \end{aligned} \tag{57}$$

and can be simplified to the final form in Theorem 2 using simple matrix manipulations.

$$\mathcal{G}_m\left[\mathbf{V}(\cdot)\right] \equiv \mathcal{G}\left\{\mathbf{M}(\Theta_\imath,\bar{H}_\imath),\mathbf{V}(\cdot),\mathcal{G}\left[\mathbf{W}_{\mathbf{x}}(H_\imath,I_\imath),\mathbf{M}(\Theta_\imath,\bar{H}_\imath),\partial_{\mathbf{W}_{\mathbf{x}}}e(H_\imath,I_\imath)\right]\right\} + \mathcal{G}\left\{\mathbf{M}(\Theta_\imath,H_\imath),\mathbf{V}(\cdot),\mathcal{G}\left[\mathbf{M}(\Theta_\imath,H_\imath)^{-1},\mathbf{M}(\Theta_\imath,H_\imath),\mathcal{G}\left(\mathbf{W}_{\mathbf{x}}(H_\imath,I_\imath),\mathbf{M}(\Theta_\imath,H_\imath)^{-1},\partial_{\mathbf{W}_{\mathbf{x}}}e(H_\imath,I_\imath)\right)\right]\right\} \quad (60)$$

The computation of the adjoined error gradient with respect to the STM output weights is more involved because these weights appear in both systems in (17). Let $\mathbf{M}(\Theta_i,H_l) \equiv \mathbf{V}_d(\Theta_i,H_l)\mathbf{\Lambda}(H_l)$, such that the input-weight constraint can be written as

$$\mathbf{W}_{\mathbf{x}}(H_\imath,I_\imath) = \mathbf{M}(\Theta_\imath,H_\imath)^{-1}\cdot\left[\mathbf{K}_{\imath\imath} - \mathbf{M}(\Theta_\imath,\bar{H}_\imath)\mathbf{W}_{\mathbf{x}}(\bar{H}_\imath,I_\imath)\right]. \quad (58)$$

Then, because $\mathbf{\Lambda}$ is a constant matrix, $\mathbf{M}(\cdot)$ can be used to refer to the influence of selected output weights on the above constraint. By applying the chain rule (8), the adjoined gradient can be computed with respect to the STM output weights as

$$\frac{\partial E}{\partial \mathbf{V}}(\Theta_\imath,\bar{H}_\imath) = \frac{\partial e}{\partial \mathbf{V}}(\Theta_\imath,\bar{H}_\imath) + \mathcal{G}\left[\mathbf{V}(\Theta_\imath,\bar{H}_\imath),\mathbf{V}(\Theta_\imath,H_\imath),\partial_{\mathbf{V}}e(\Theta_\imath,H_\imath)\right] + \sum_{\imath}\mathcal{G}\left[\mathbf{V}(\Theta_\imath,\bar{H}_\imath),\mathbf{W}_{\mathbf{x}}(H_\imath,I_\imath),\partial_{\mathbf{W}_{\mathbf{x}}}e(H_\imath,I_\imath)\right]. \quad (59)$$

While the first two terms are already written in terms of $\partial_{\mathbf{V}}e$, the third term must be expanded using the *recursion property* in order to express it with respect to $\partial_{\mathbf{W}_{\mathbf{x}}}e$. For simplicity, this expansion is written with respect to the operator as shown in (60) at the top of the page, which represents an intermediate differentiation step arising from applying the chain rule recursively to multiple constraints. With this notation, the term in the summation in (59) can be expanded as

$$\mathcal{G}\left[\mathbf{V}(\Theta_\imath,\bar{H}_\imath),\mathbf{W}_{\mathbf{x}}(H_\imath,I_\imath),\partial_{\mathbf{W}_{\mathbf{x}}}e(H_\imath,I_\imath)\right] = \mathcal{G}\left\{\mathbf{V}(\Theta_\imath,H_\imath),\mathbf{V}(\Theta_\imath,\bar{H}_\imath),\mathcal{G}_m\left[\mathbf{V}(\Theta_\imath,H_\imath)\right]\right\} + \mathcal{G}_m\left[\mathbf{V}(\Theta_\imath,\bar{H}_\imath)\right] \quad (61)$$

and the adjoined error gradient can be written as

$$\frac{\partial E}{\partial \mathbf{V}}(\Theta_\imath,\bar{H}_\imath) = \partial_{\mathbf{V}}e(\Theta_\imath,\bar{H}_\imath) + \mathcal{G}_m\left[\mathbf{V}(\Theta_\imath,\bar{H}_\imath)\right] + \left\{\partial_{\mathbf{V}}e(\Theta_\imath,H_\imath) + \mathcal{G}_m\left[\mathbf{V}(\Theta_\imath,H_\imath)\right]\right\}\cdot\left[\mathbf{\Sigma}(\bar{H}_\imath)\mathbf{\Sigma}(H_\imath)^{-1}\right]^T, \qquad \imath = 1,2. \quad (62)$$

The adjoined error gradient for the auxiliary weights and the output bias can be computed using the same approach, as shown in [45]. Because the matrix $\mathbf{V}_d(\cdot)\mathbf{\Lambda}(\cdot)$ in (17) is the product of a diagonal and a block-diagonal matrix, it is invertible provided that the diagonal elements are nonzero. Because the diagonal elements consist of nonzero output weights and of sigmoidal derivatives that are always greater than zero, then this matrix is always invertible.

REFERENCES

[1] K. Yamauchi, N. Yamaguchi, and N. Ishii, "Incremental learning methods with retrieving of interfered patterns," *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1351–1365, Nov. 1999.

[2] H. Yamakawa, D . Masumoto, T. Kimoto, and S. Nagata, "Active data selection and subsequent revision for sequential learning," (in Japanese) Tech. Rep. IEICE, NC92/99, 1993.

[3] M. Kobayashi, A. Zamani, S. Ozawa, and S. Abe, "Reducing computations in incremental learning for feedforward neural network with long-term memory," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2001, pp. 1989–1994.

[4] J. Mándziuk and L. Shastri, "Incremental class learning—An approach to long life and scalable learning," in *Int. Joint Conf. Neural Netw.*, Jul. 1999, pp. 1319–1324.

[5] M. Kotani, K. Akazawa, S. Ozawa, and H. Matsumoto, "Detection of leakage sound by using modular neural networks," in *Proc. 16th Congr. Int. Meas. Confederat.*, 2000, pp. 347–351.

[6] R. Lamprecht and J. LeDoux, "Structural plasticity and memory," *Nature Rev.—Neurosci.*, vol. 5, pp. 45–54, Jan. 2004.

[7] J. Si, A. G. Barto, W. B. Powell, and D. W. Wunsch, II, *Handbook of Learning and Approximate Dynamic Programming*. New York: Wiley, 2004.

[8] G. K. Venayagamoorthy, R. G. Harley, and D. C. Wunsch, "Implementation of adaptive critic-based neurocontrollers for turbogenerators in a multimachine power system," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1047–1064, Sep. 2003.

[9] V. Yadav, R. Padhi, and S. N. Balakrishnan, "Robust/optimal temperature profile control of a high-speed aerospace vehicle using neural networks," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 1115–1128, Jul. 2007.

[10] D. Liu, Y. Zhang, and H. Zhang, "A self-learning call admission control scheme for CDMA cellular networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 5, pp. 1219–1228, Sep. 2005.

[11] R. Enns and J. Si, "Helicopter trimming and tracking control using direct neural dynamic programming," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 929–939, Jul. 2003.

[12] G. G. Lendaris and J. C. Neidhoefer, "Guidance in the use of adaptive critics for control," in *Learning and Approximate Dynamic Programming*, J. Si, A. Barto, and W. Powell, Eds. New York: Wiley, 2004, p. 114.

[13] M. Agarwal, "Combining neural and conventional paradigms for modelling, prediction and control," *Int. J. Syst. Sci.*, vol. 28, no. 1, pp. 65–81, 1997.

[14] M. M. Polycarpou, "Stable adaptive neural control scheme for nonlinear systems," *IEEE Trans. Autom. Control*, vol. 14, no. 3, pp. 447–451, 1996.

[15] S. Ferrari and R. F. Stengel, "Model-based adaptive critic designs," in *Learning and Approximate Dynamic Programming*, J. Si, A. Barto, and W. Powell, Eds. New York: Wiley, 2004.

[16] S. Ferrari and R. F. Stengel, "Classical/neural synthesis of nonlinear control systems," *J. Guid. Control Dyn.*, vol. 25, no. 3, pp. 442–448, 2002.

[17] S. Ferrari and R. F. Stengel, "On-line adaptive critic flight control," *J. Guid. Control Dyn.*, vol. 27, no. 5, pp. 777–786, 2004.

[18] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends Cogn. Sci.*, vol. 3, no. 4, pp. 128–135, 1999.

[19] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[20] M. Reidmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Netw.*, San Francisco, CA, 1993, pp. 586–591.

[21] D. P. Bertsekas, "A new class of incremental gradient methods for least squares problems," *SIAM J. Optim.*, vol. 7, pp. 913–926, 1997.

[22] M. V. Solodov, "Incremental gradient algorithms with stepsizes bounded away from zero," *Comput. Optim. Appl.*, vol. 11, pp. 23–35, 1998.

[23] D. L. Wang and B. Yuwono, "Incremental learning of complex temporal patterns," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1465–1481, Nov. 1996.

[24] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, pp. 71–99, 1993.

[25] F. Flentge, "Locally weighted interpolating growing neural gas," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1382–1393, Nov. 2006.

[26] C. Constantinopoulos and A. Likas, "An incremental training method for the probabilistic RBF network," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 966–974, Jul. 2006.
[27] S. Wan and L. E. Banta, "Parameter incremental learning algorithm for neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1424–1438, Nov. 2006.
[28] R. F. Stengel, *Optimal Control and Estimation*. New York: Dover, 1994, pp. 36–41.
[29] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Belmont, MA: Athena Scientific, 1996.
[30] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 24–38, Jan. 2005.
[31] S. Limanons and J. Si, "Neural network-based control design: An LMI approach," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1422–1429, 1998.
[32] Y. C. Chu and K. Glover, "Stabilization and performance synthesis for systems with repeated scalar nonlinearities," *IEEE Trans. Autom. Control*, vol. 44, no. 3, pp. 484–496, Mar. 1999.
[33] J. Sjöberg, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.
[34] O. L. Mangasarian and E. W. Wild, "Nonlinear knowledge in kernel approximation," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 300–306, Jan. 2007.
[35] R. F. Stengel, *Flight Dynamics*. Princeton, NJ: Princeton Univ. Press, 2004.
[36] R. F. Stengel and C. Marrison, "Design of robust control systems for hypersonic aircraft," *J. Guid. Control Dyn.*, vol. 21, no. 1, pp. 58–63, 1997.
[37] C. Huang and R. F. Stengel, "Restructurable control using proportional-integral model following," *J. Guid. Control Dyn.*, vol. 13, no. 2, pp. 303–309, 1990.
[38] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[39] R. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.
[40] S. N. Balakrishnan and V. Biega, "Adaptive-critic-based neural networks for aircraft optimal control," *J. Guid. Control Dyn.*, vol. 19, no. 4, pp. 856–898, 1996.
[41] J. Kalviste, "Spherical mapping and analysis of aircraft angles for maneuvering flight," *J. Aircraft*, vol. 24, no. 8, pp. 523–530.
[42] D. S. Bernstein, *Matrix Mathematics*. Princeton, NJ: Princeton Univ. Press, 2005.
[43] J. R. Magnus and H. Neudecker, *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Chichester, U.K.: Wiley, 1988.
[44] R. Mathias, "Chain rule for matrix functions and applications," *SIAM J. Matrix Anal. Appl.*, vol. 17, pp. 610–620, 1996.
[45] M. Jensenius, "Constrained learning in neural control systems," M.S. thesis, Dept. Mech. Eng., Duke Univ., Durham, NC, 2005.

Silvia Ferrari (S'01–M'02) received the B.S. degree from Embry-Riddle Aeronautical University, Daytona Beach, FL, in 1997, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, in 1999 and 2002, respectively.

Currently, she is an Assistant Professor of Mechanical Engineering and Materials Science at Duke University, Durham, NC, where she directs the Laboratory for Intelligent Systems and Controls (LISC). Her principal research interests include robust adaptive control of aircraft, learning and approximate dynamic programming, and control distributed sensor networks.

Dr. Ferrari is a member of the American Society of Mechanical Engineers (ASME), The International Society for Optical Engineers (SPIE), and the American Institute of Aeronautics and Astronautics (AIAA). She is the recipient of the U.S. Office of Naval Research (ONR) Young Investigator award (2004), the national Science Foundation (NSF) CAREER award (2005), and the Presidential Early Career Award for Scientists and Engineers (PECASE) award (2006).

Mark Jensenius (M'07) received the B.S. degree in aeronautical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2003, and the M.S. degree in mechanical engineering from Duke University, Durham, NC, in 2005.

Currently, he is a Systems Engineer in the Guidance, Navigation, and Control group, Air and Missile Defense Department, Applied Physics Laboratory, The John Hopkins University, Laurel, MD. His research is currently focused on divert and attitude control systems for exo–atmospheric vehicles.

Mr. Jensenius is the recipient of the James B. Duke Fellowship Award (2003 and 2004), the Rickett's Prize (2003), and the Leopold L. Balleisen Prize (2003).